SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

Parallel computation methods on CPU architectures

Claudia Blaas-Schenner
VSC Research Center, TU Wien

VIENNA SCIENTIFIC CLUSTER

01/2022

Univerza v Ljubljani

TU WIEN TECHNISCHE UNIVERSITÄT WIEN

CINECA

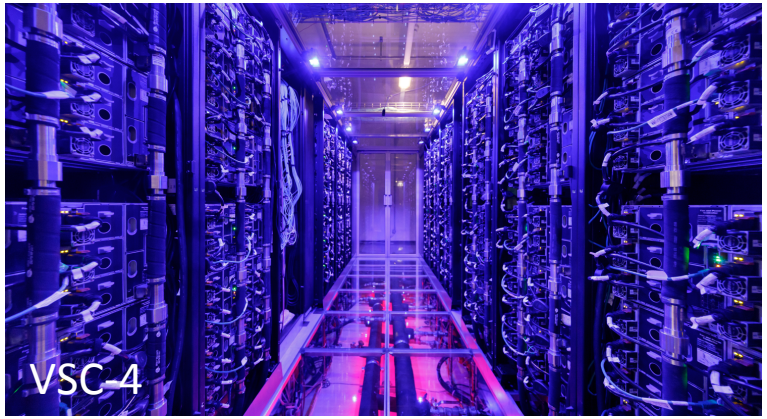
VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

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

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

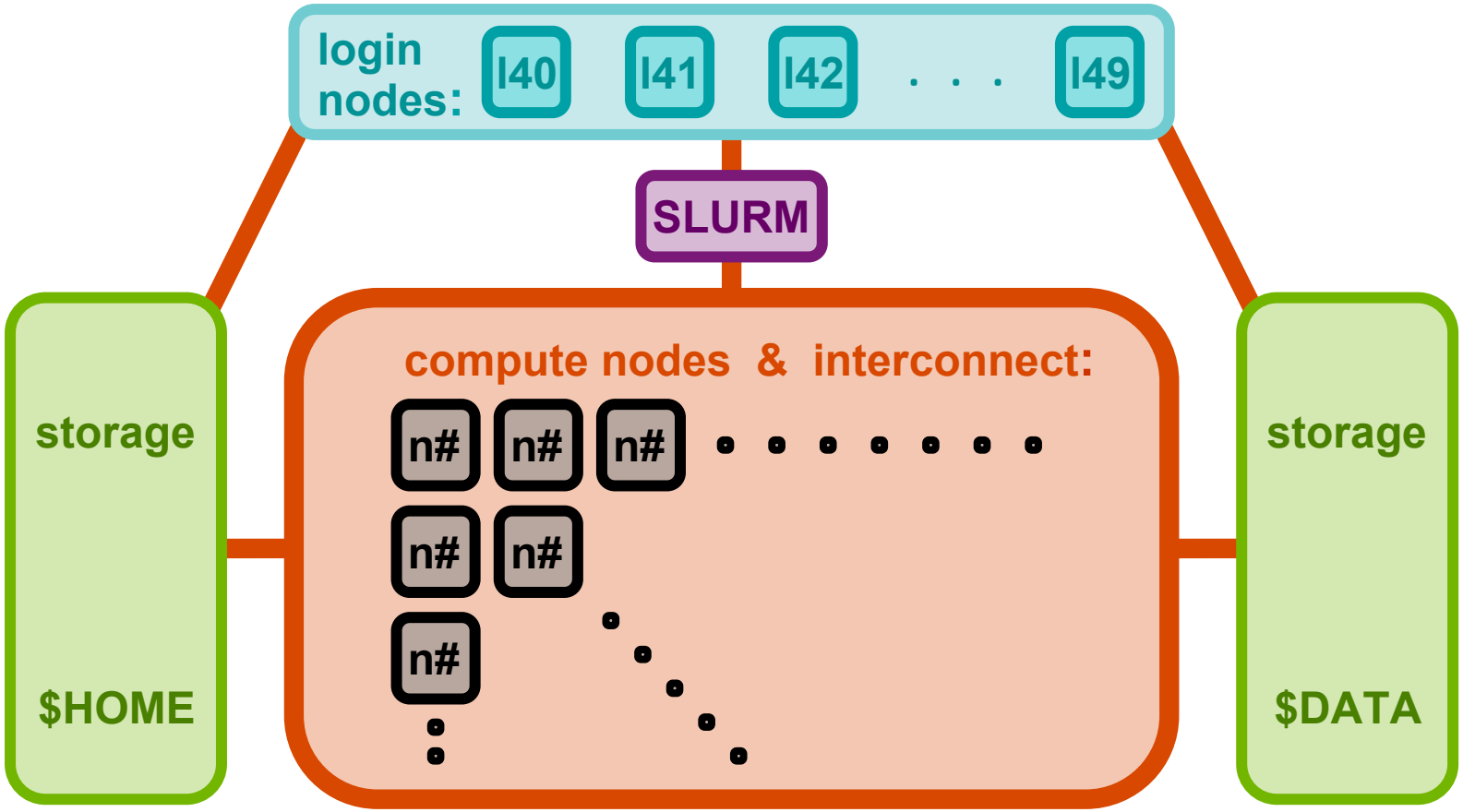


VSC-5

soon to come…

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

components of HPC clusters



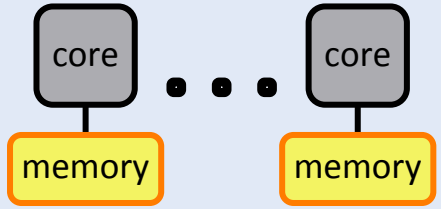
login nodes

compute nodes

shared (login, storage)

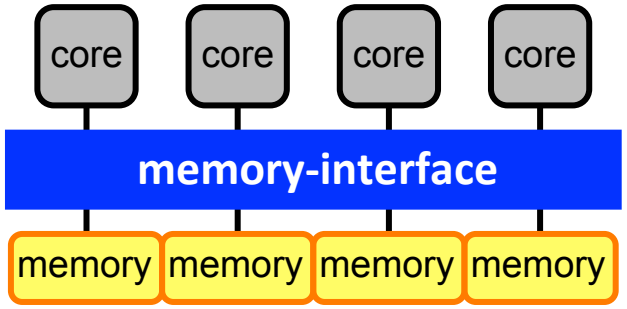
user exclusive (compute nodes)

parallel hardware



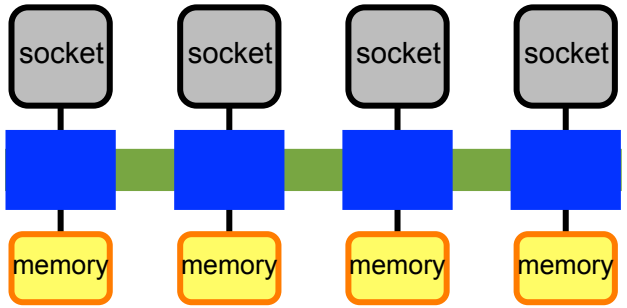
SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

shared memory



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

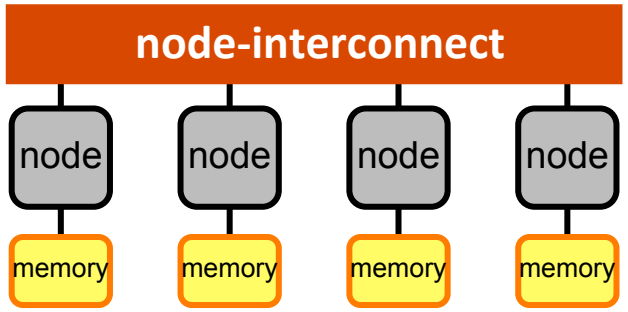
socket / CPU



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

node

distributed memory



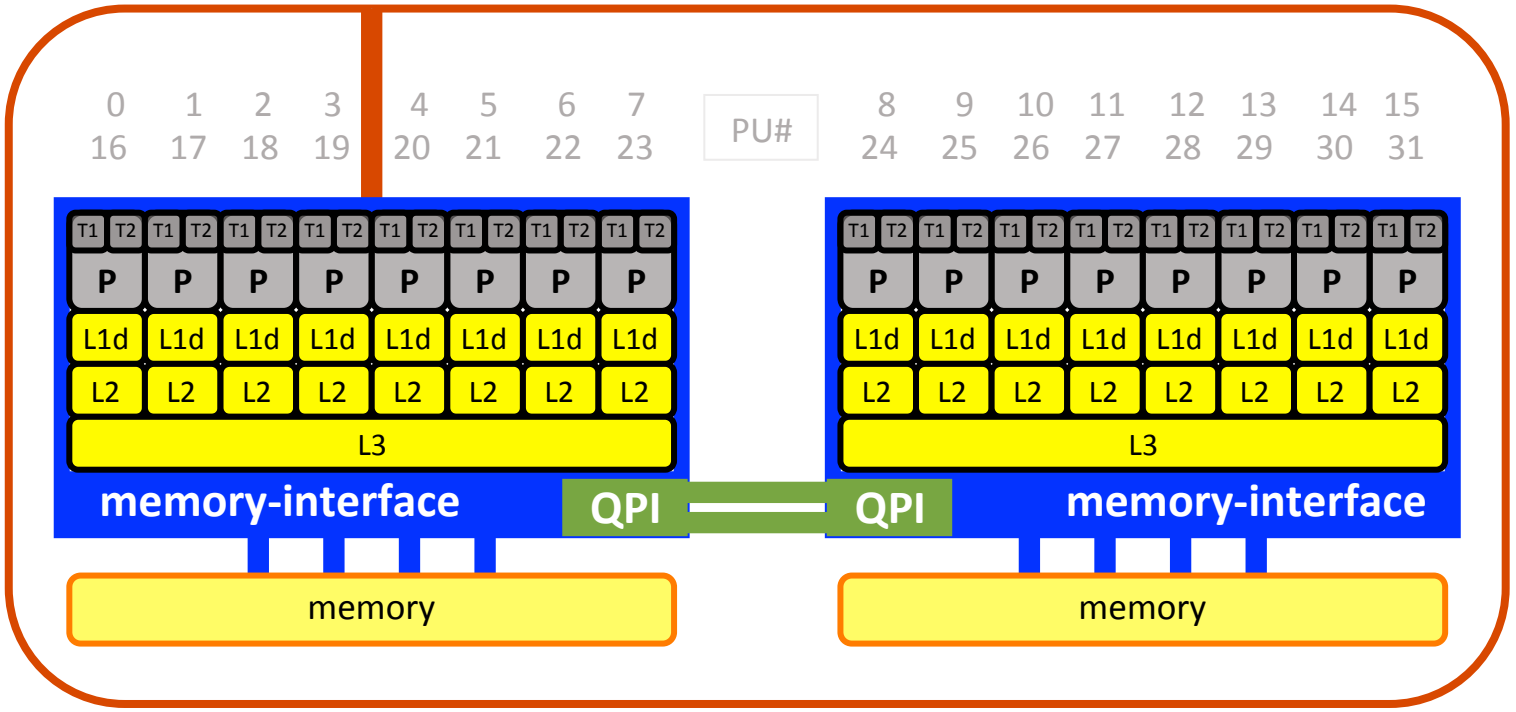
cluster: → **node-interconnect**
NUMA (non-uniform memory access)
! fast access only to its own memory !

cluster

shared memory programming with **OpenMP**

MPI works everywhere

compute nodes



example:

1 node

2 sockets (CPUs)

8/10 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

schematic figure:

3-level fat tree

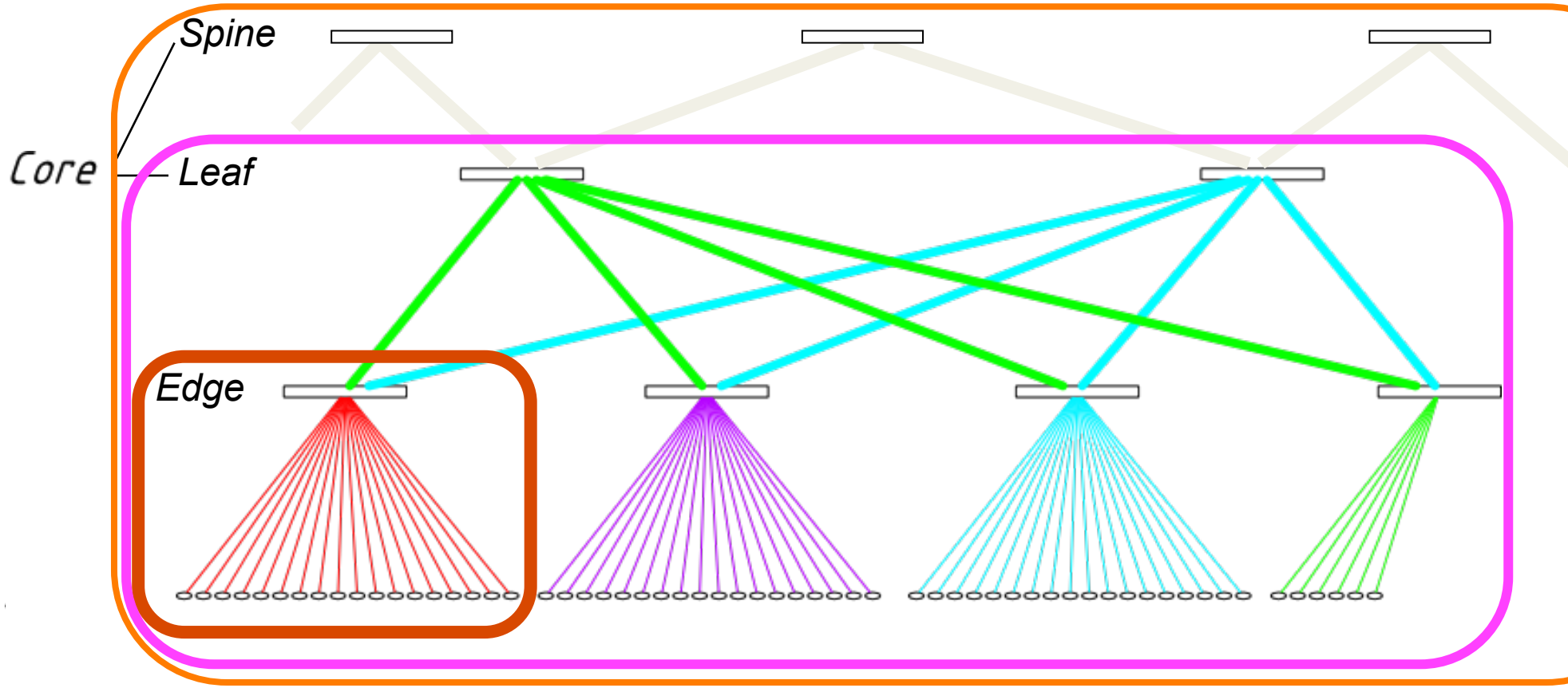
2-level fat-tree

1st level switches

compute nodes

attached to the

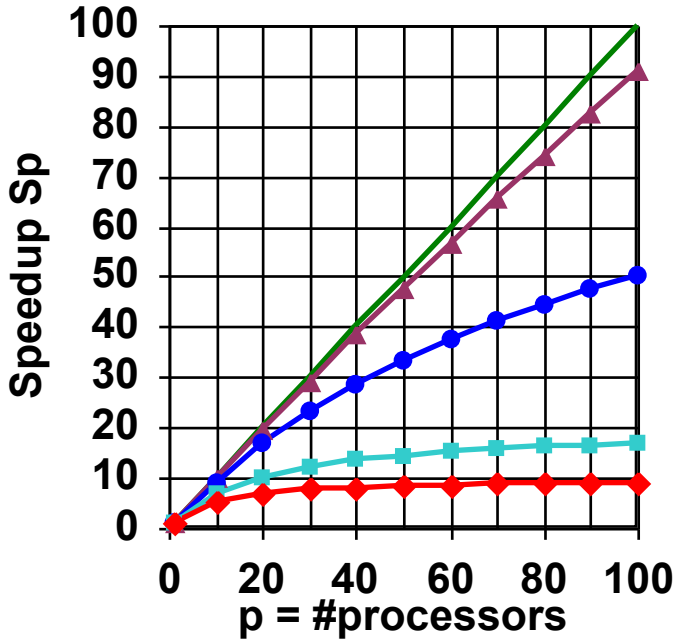
lowest level

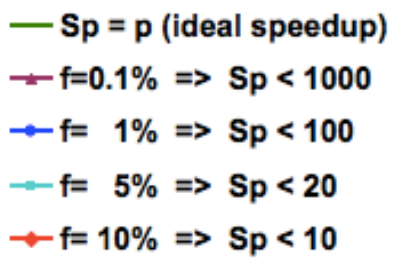


Amdahl's Law

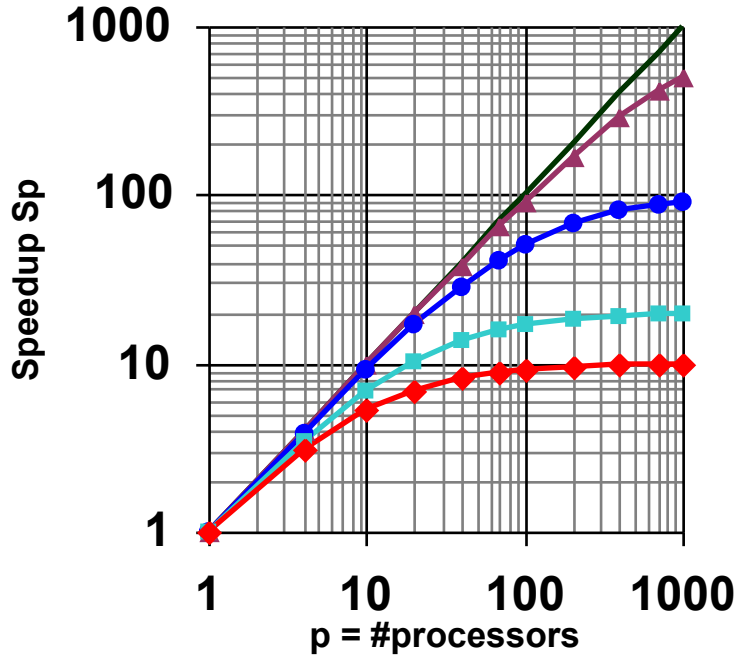
$T_{parallel,\,p} = f \cdot T_{serial} + (1-f) \cdot T_{serial} / p$ f ... sequential part of code **neglecting time for communication**

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



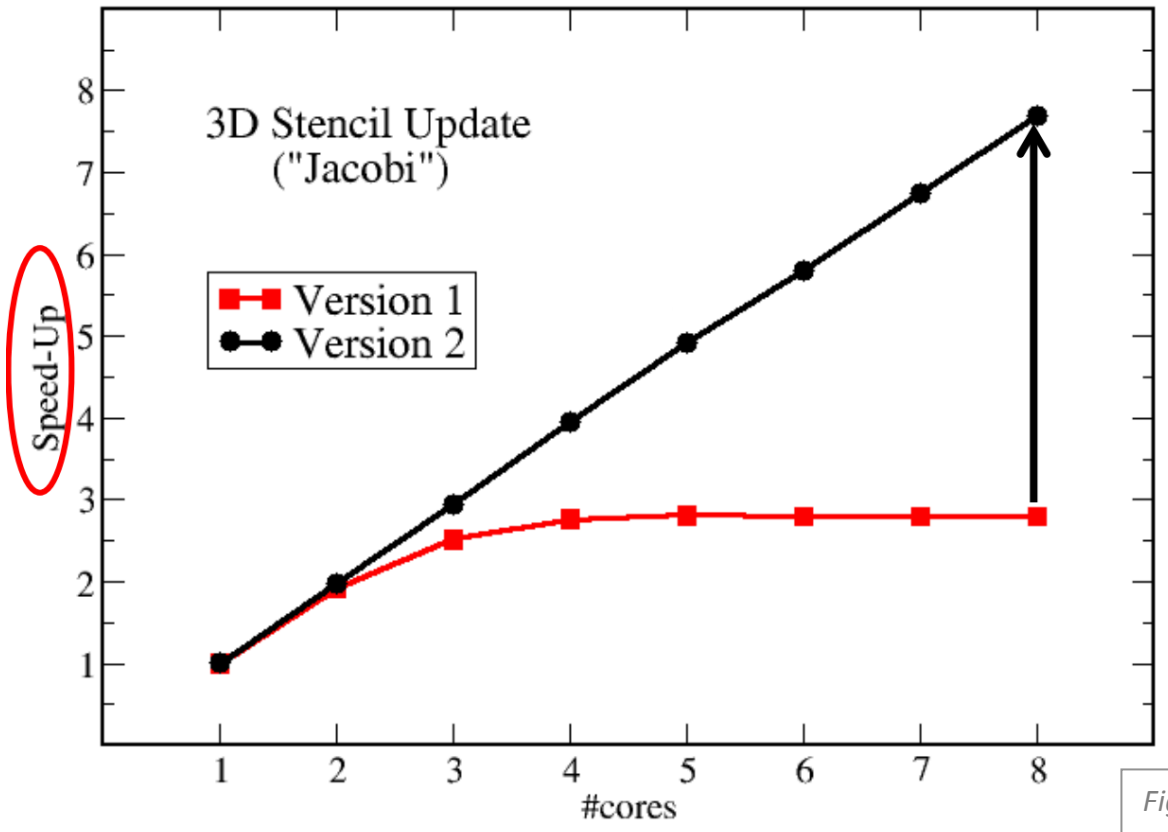


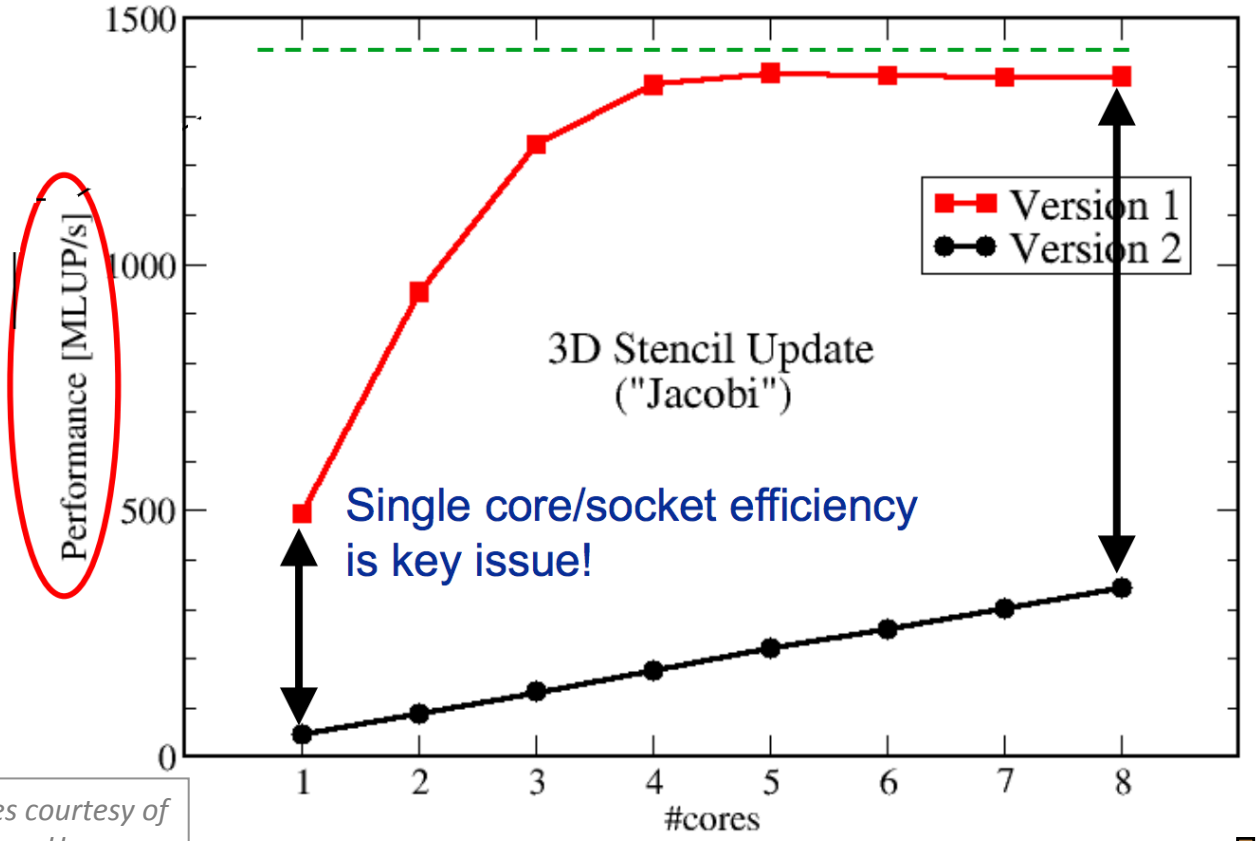
Figures courtesy of Rolf Rabenseifner.



Speedup = ratio – no absolute performance !

scalability vs. performance



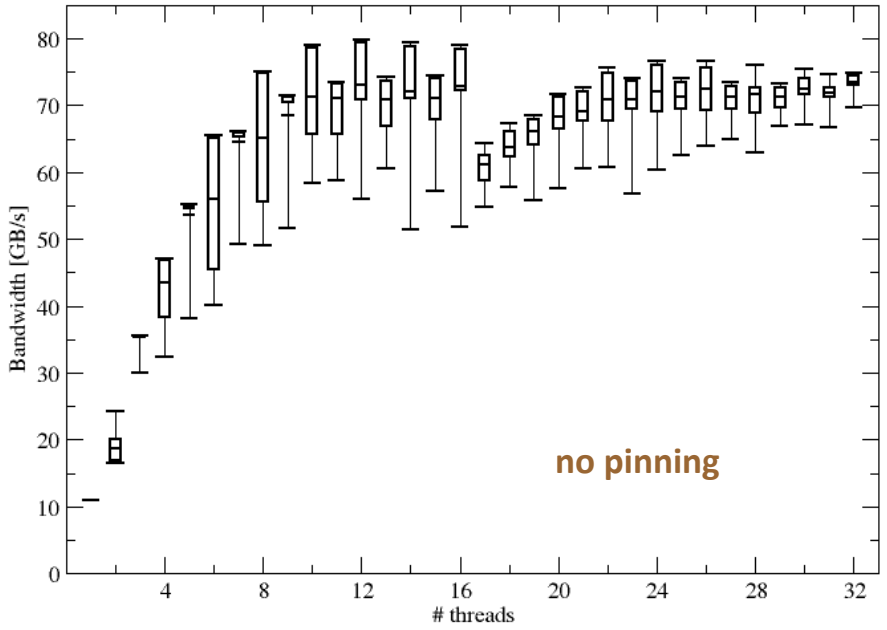


Figures courtesy of Georg Hager.

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))
```

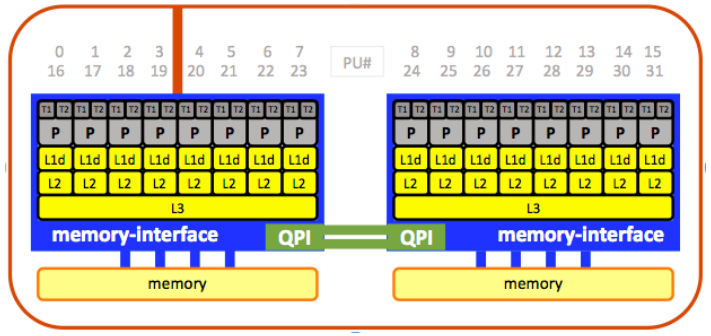
pinning ?

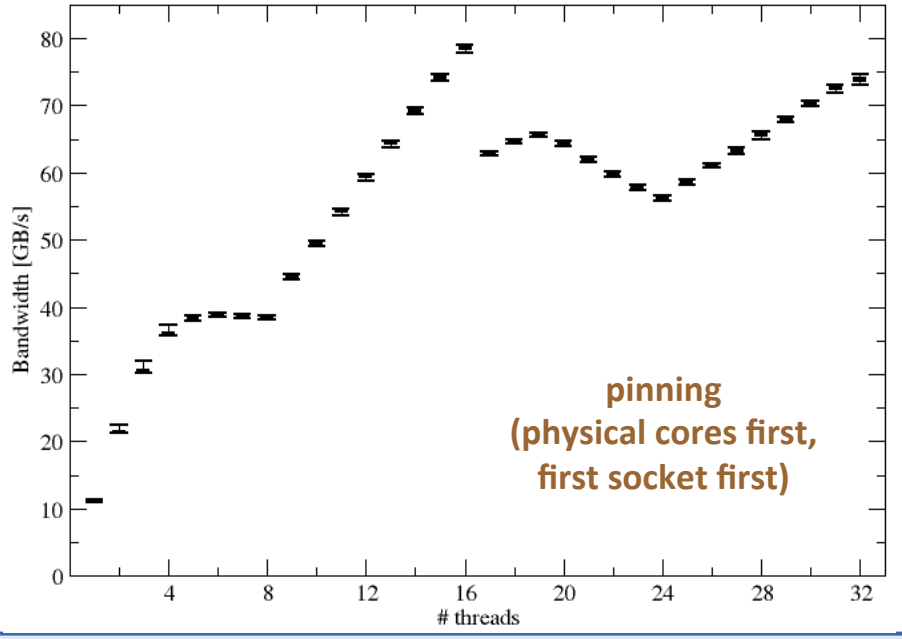


OpenMP
STREAM benchmark

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

HPC

computation *exclusive*
node / core

communication *exclusive*
message passing *(BF)*

I/O *shared*
parallel FS *with all users*

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)

modules & compiling

- **module environment** [spack]

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

- **compiling with GCC**

```
module load gcc…

cc --version
mpicc --version

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

- **compiling with Intel**

```
module load intel…

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

- **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


source ~/.bashrc
```

SLURM job script (pure MPI)


```
#!/bin/bash                          # → @vsc3

#SBATCH -J test                      # SLURM_JOB_NAME
#SBATCH -N 2                         # SLURM_JOB_NUM_NODES
#SBATCH --tasks-per-node=20          # SLURM_NTASKS_PER_NODE  [1 mpi/core]

# <do_my_work>
module purge                         # recommended to be done in all jobs !!!!!
module load intel/18                 # load only modules actually needed by job

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

demo HPC literacy

- **make yourself familiar with: login, modules & compiling, job submission**
- `cp -a ~/00/HPC .` → copy the HPC exercises
- `cd HPC` → go to the folder

`module load intel/18` → @vsc3

`export MPI_PROCESSES=4`

`export OMP_NUM_THREADS=5`

→ **`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/

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.