VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

INTRODUCTION TO HIGH PERFORMANCE COMPUTING

PARALLEL PROGRAMMING BASICS

Ondřej Meca

PROGRAMMING PARALLEL MACHINES

Repository with examples:

- https://code.it4i.cz/training/intro2hpc
- /mnt/proj2/dd-24-88/intro2hpc/4_parallel_programming

- src/mapping.cpp: print mapping of your allocation
- src/threaded.cpp: hybrid application for testing

- sh make.sh: compile examples into build directory

- jobs/...: directory with Slurm script examples

PROGRAMMING PARALLEL MACHINES

Sequential (serial) programs:

- operate on similar principles everywhere

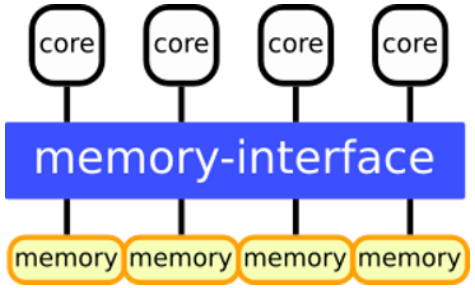
Parallel programs:

- dependent on the target parallel architecture
- more difficult to write than sequential ones
  - several new classes of software bugs (e.g., race conditions)
  - difficult debugging
- more difficult to run efficiently
  - mapping, pinning

PARALLEL HARDWARE

Multi-processor (socket, CCD in the case of AMD processors)

- all cores share the same memory
- single / global address space
- the same speed to all memory locations (uniform memory access)



socket

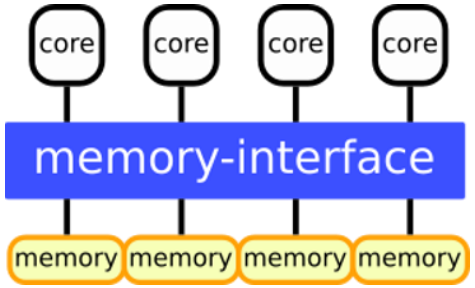
UMA (uniform memory access)

SMP (symmetric multi-processing)

PARALLEL HARDWARE

Several sockets with multi-processors (node)

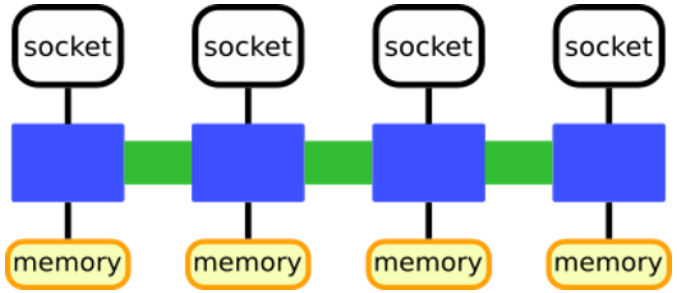
- memory is shared among all CPUs
- single / global address space
- the same speed to all memory locations (uniform memory access)?



socket

UMA (uniform memory access)

SMP (symmetric multi-processing)



node

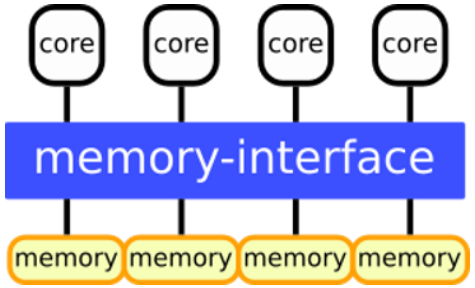
ccNUMA (cache-coherent non-uniform ...)

first touch, pinning!

PARALLEL HARDWARE

Several sockets with multi-processors (node)

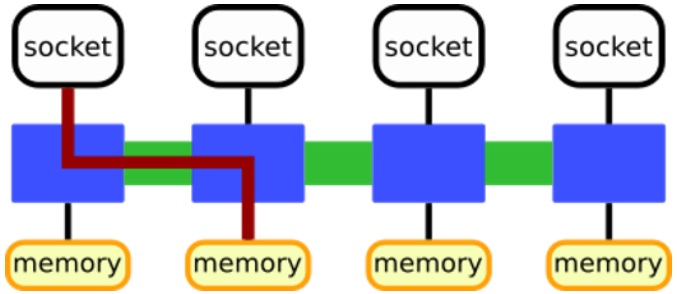
- memory is shared among all CPUs
- single / global address space
- ~~the same speed to all memory locations (uniform memory access)?~~
- the speed is dependent on a memory location (non-uniform memory access)



socket

UMA (uniform memory access)

SMP (symmetric multi-processing)



node

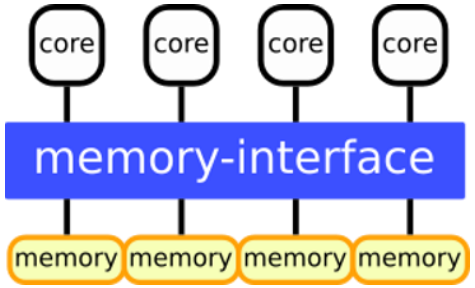
ccNUMA (cache-coherent non-uniform ...)

first touch, pinning!

PARALLEL HARDWARE

Multi-computers with various architectures (cluster)

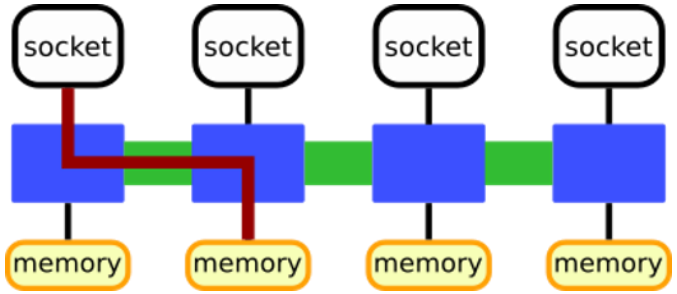
- set of nodes interconnected by a network
- each node has separated memory
- slower access to memories of other processors
- accelerated nodes



socket

UMA (uniform memory access)

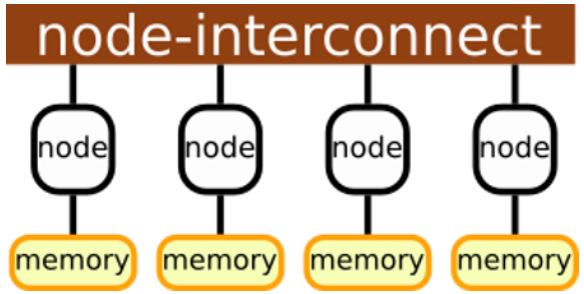
SMP (symmetric multi-processing)



node

ccNUMA (cache-coherent non-uniform ...)

first touch, pinning!



cluster

NUMA (non-uniform memory access)

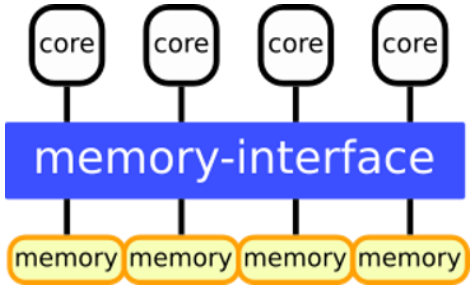
fast access to own memory only

PARALLEL HARDWARE

OpenMP: shared memory (socket, node)

MPI: distributed memory (socket, node, cluster)

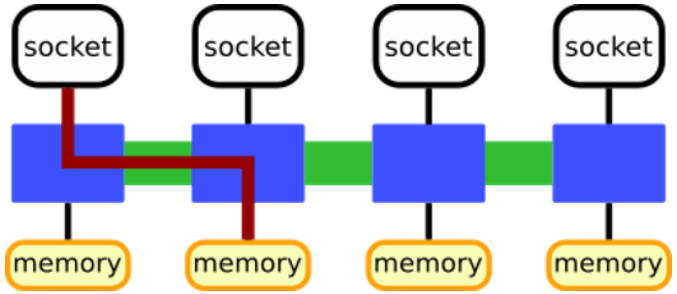
CUDA: accelerated nodes



socket

UMA (uniform memory access)

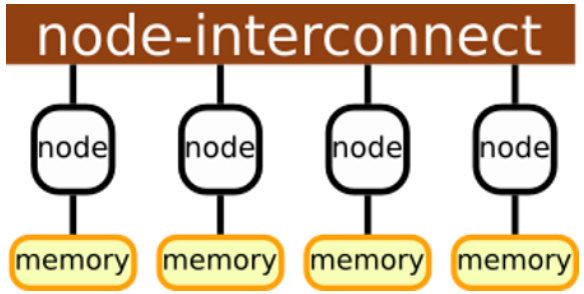
SMP (symmetric multi-processing)



node

ccNUMA (cache-coherent non-uniform ...)

first touch, pinning!



cluster

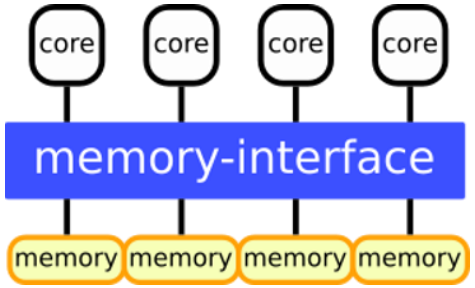
NUMA (non-uniform memory access)

fast access to own memory only

PARALLEL HARDWARE

Hybrid approach

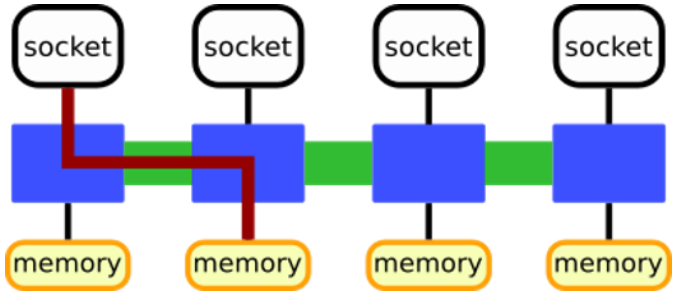
- combination of more approaches (OpenMP, MPI, CUDA,...)
- potential to fully utilize current (future) hardware



socket

UMA (uniform memory access)

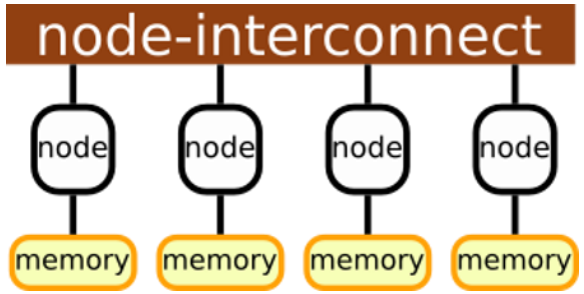
SMP (symmetric multi-processing)



node

ccNUMA (cache-coherent non-uniform ...)

first touch, pinning!



cluster

NUMA (non-uniform memory access)

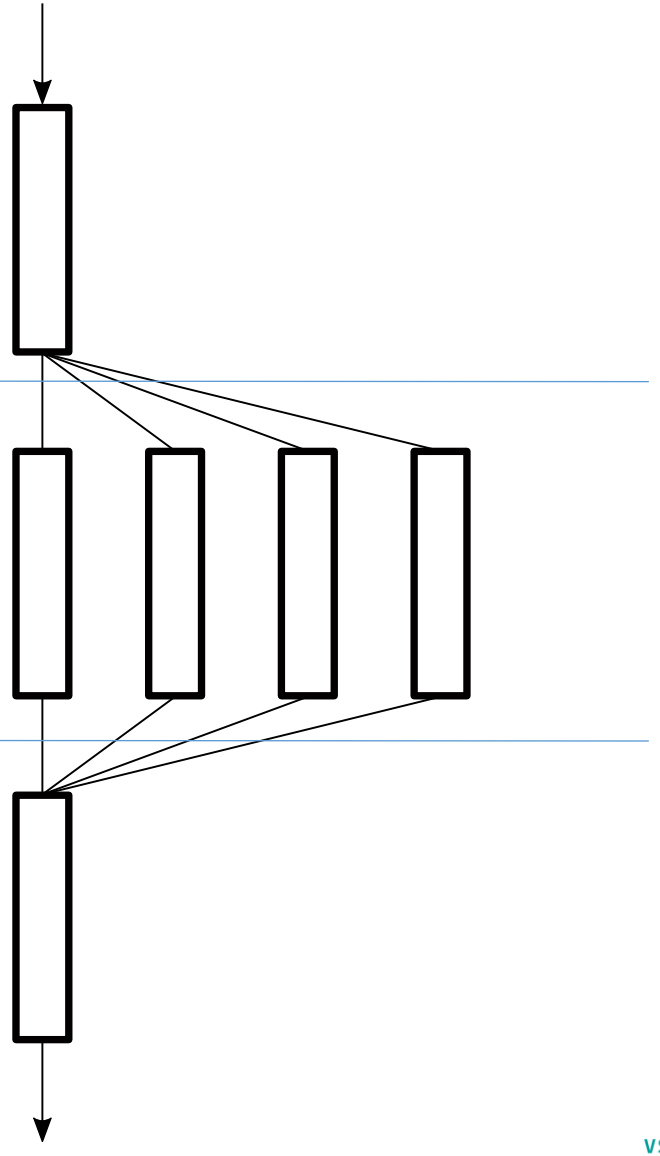
fast access to own memory only

OPENMP

- **Open M**ulti-**P**rocessing
  - API for writing portable multi-threaded applications based on the shared variables model with interfaces for Fortran, C, and C++
  - compilers available on most platforms (Unix, Windows, etc.)
- A set of compiler directives, library routines and environment variables
- A standard developed by the OpenMP Architecture Review Board
  - http://www.openmp.org
  - first specification in 1997, current version 6.0
- No data distribution, no communication (threads communicate via shared variables)
- Allows incremental parallelization
  - i.e., the sequential program evolves into a parallel program
  - single source code for both the sequential and parallel versions

OPENMP

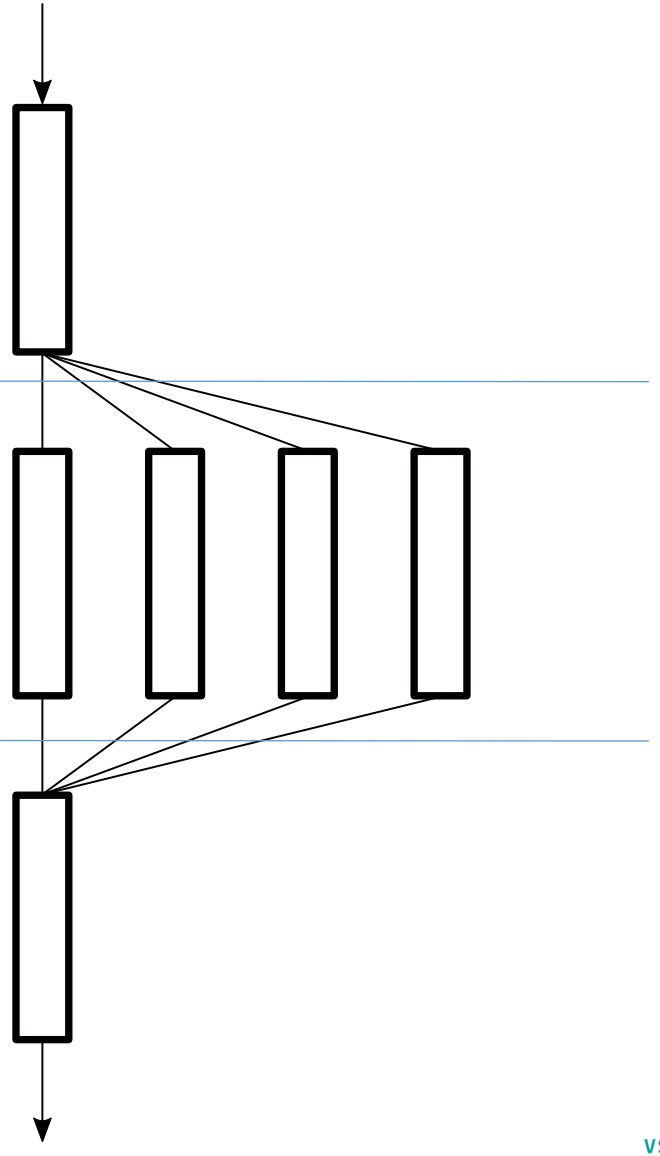
```
#include "omp.h"

int main(int argc, char **argv) {
  int iam = 0, np = 1;

  #pragma omp parallel private(iam, np) /* Parallel region */
  {
    #if defined (_OPENMP)
      np = omp_get_num_threads();
      iam = omp_get_thread_num();
    #endif
    printf("Hello from thread %d out of %d\n", iam, np);
  }
}
```



```
$ g++ -fopenmp hello.cpp -o hello
$ OMP_NUM_THREADS=4 ./hello

Hello from thread 2 out of 4
Hello from thread 0 out of 4
Hello from thread 1 out of 4
Hello from thread 3 out of 4
```

OPENMP

```
#include "omp.h"

int main(int argc, char **argv) {
  int iam = 0, np = 1;

  #pragma omp parallel private(iam, np) /* Parallel region */
  {
    #if defined (_OPENMP)
      np = omp_get_num_threads();
      iam = omp_get_thread_num();
    #endif
    printf("Hello from thread %d out of %d\n", iam, np);
  }
}
```



```
$ g++ -fopenmp hello.cpp -o hello
$ OMP_NUM_THREADS=4 ./hello

Hello from thread 2 out of 4
Hello from thread 0 out of 4
Hello from thread 1 out of 4
Hello from thread 3 out of 4
```

OPENMP

- Mainly directives applied to the following block of code

```
#pragma omp parallel [ clause [ [ , ] clause ] ... ] new-line
{
        // code performed by all threads
}
```

- Clauses:
  - private (list), shared (list),
  - reduction (operator: list), schedule (type [, chunk])
- Synchronization:
  - master, critical, atomic, barrier
- Environment variables:
  - OMP_NUM_THREADS, OMP_PLACES, OMP_PROC_BIND

- https://www.openmp.org/resources/tutorials-articles/

OPENMP

- Mainly directives applied to the following block of code

```
#pragma omp parallel [ clause [ [ , ] clause ] ... ] new-line
{
        // code performed by all threads
}
```

- Clauses:
  - private (list), shared (list),
  - reduction (operator: list), schedule (type [, chunk])
- Synchronization:
  - master, critical, atomic, barrier
- **Environment variables:**
  - **OMP_NUM_THREADS, OMP_PLACES, OMP_PROC_BIND**

- https://www.openmp.org/resources/tutorials-articles/

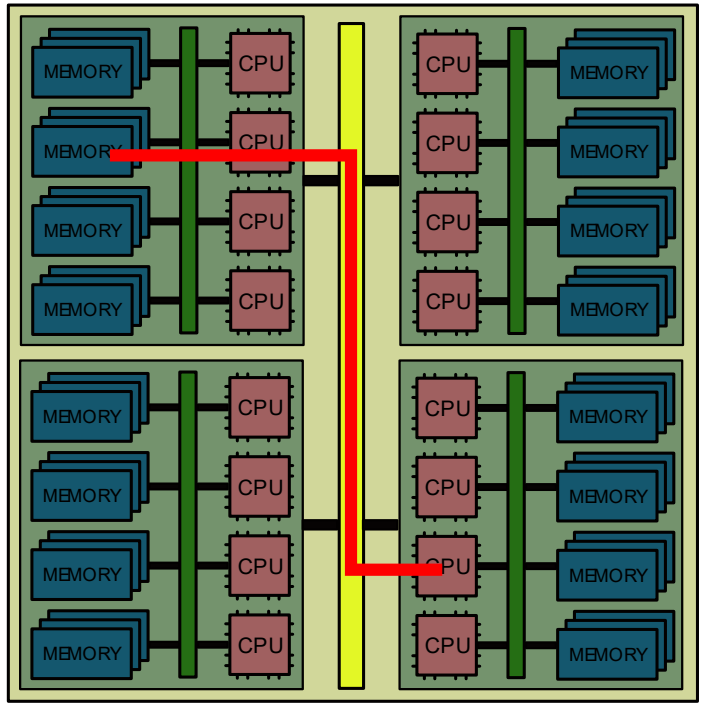
OPENMP PITFALLS

- Race conditions
  - output is dependent on the detailed timing of concurrent operations
  - e.g., modifying the same variable by two threads
- False sharing
  - significant slowdown in a parallel block of code
  - e.g., modification the same cache line by two threads
- Deadlocks
  - waiting for resources that will never be available

- Sequential equivalence:
  - strong: bitwise identical results
  - weak: mathematically equivalent (not bitwise identical due to the floating-point arithmetic)

OPENMP PITFALLS

- Cache coherent distributed memory (ccNUMA)
  - threads requests memory that was firstly touched by a thread from another sockets
  - the same memory should be accessed by the same thread
  - fix threads to a particular CPUs (OMP_PROC_BIND=true ./app)



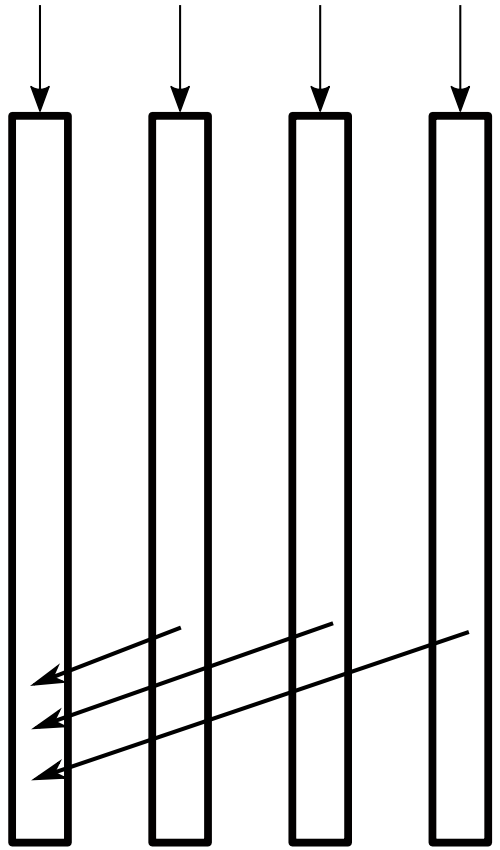

```
double *vals = new double[rows * cols];

#pragma omp parallel for collapse(2)
for (int r = 0; r < rows; ++r) {
    for (int c = 0; c < cols; ++c) {
        vals[r * cols + c] = 0;
    }
}
```

MPI

- **Message Passing Interface:**
  - standard for distributed memory parallelism with passing messages
- MPI is the interface, not a library!
  - many available libraries with an implementation (OpenMPI, mpich, Intel MPI,...)
  - some behavior is dependent on a particular implementation
- A standard developed by the MPI Forum
  - http://www.mpi-forum.org
  - first specification in 1994, current version 5.0
- Explicit definition of data distribution and communication
- MPI application is a set of processes that cooperate with each other by sending messages

MPI

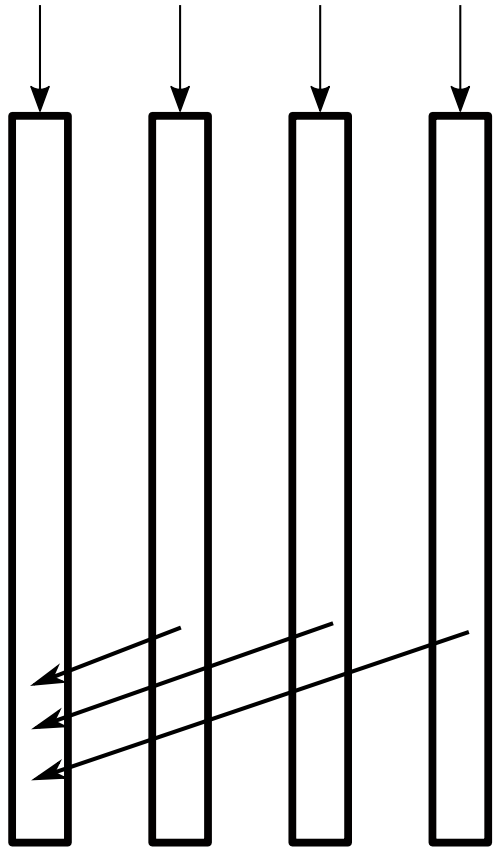
```
#include "mpi.h"

int main(int argc, char **argv) {
  int rank, size;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  printf("Hello from process %d out of %d\n", rank, size);
  if (rank==0) {
    // recv messages
  } else {
    // send a message
  }
  MPI_Finalize();
}
```



```
$ mpic++ hello.cpp -o hello
$ mpirun -n 4 ./hello

Hello from process 2 out of 4
Hello from process 0 out of 4
Hello from process 1 out of 4
Hello from process 3 out of 4
```

MPI

```
#include "mpi.h"

int main(int argc, char **argv) {
  int rank, size;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  printf("Hello from process %d out of %d\n", rank, size);
  if (rank==0) {
    // recv messages
  } else {
    // send a message
  }
  MPI_Finalize();
}
```



```
$ mpic++ hello.cpp -o hello
$ mpirun -n 4 ./hello

Hello from process 2 out of 4
Hello from process 0 out of 4
Hello from process 1 out of 4
Hello from process 3 out of 4
```

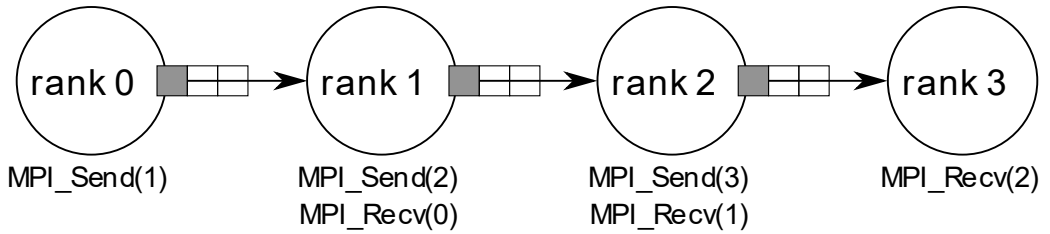
MPI PITFALLS

- Race conditions
  - output is dependent on the detailed timing sent/received operations
  - e.g., output is dependent on the order of received messages
- Deadlocks
  - waiting for resources that will never be available

- Sequential equivalence:
  - strong: bitwise identical results
  - weak: mathematically equivalent (not bitwise identical due to the floating-points arithmetic)

MPI PITFALLS

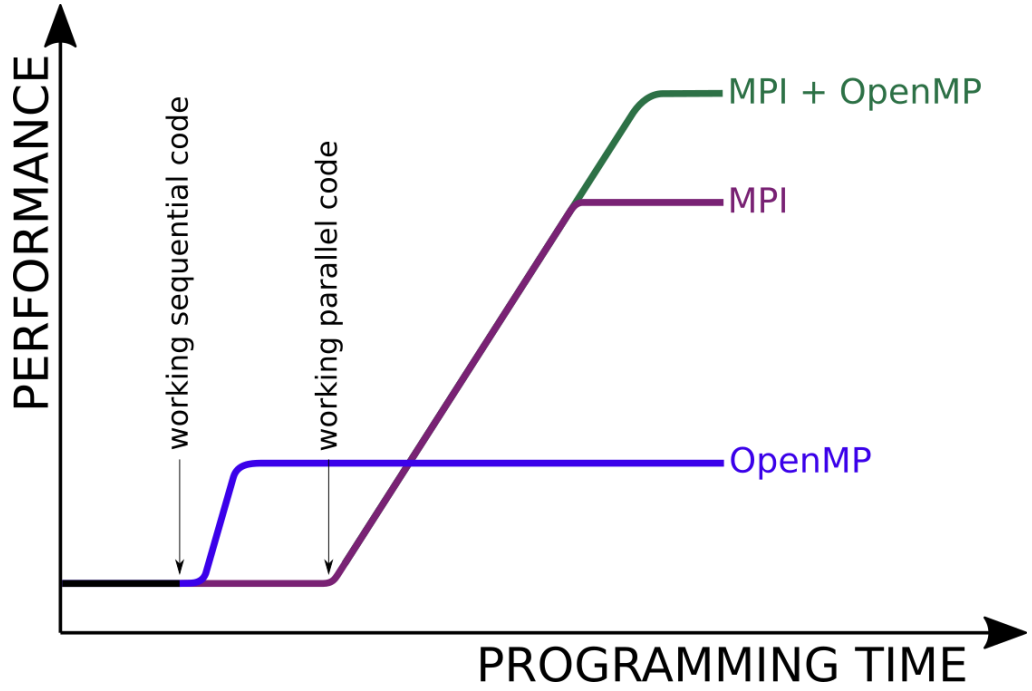
- Non-scalable functions / patterns
  - collectives with input of size *O(#processes), e.g., scatterv*
- Serialization:
  - the order of messages serializes the application
  - e.g., each process must wait to a message from the previous process
- Expensive communication
  - exchanging too much of data



- Performance is not portable
  - MPI assures only portable application!

OPENMP VS. MPI

- OpenMP
  - Incremental parallelization
- MPI:
  - usually new application with potential to fully utilize cluster capacities



PARALLEL RUN

How to run your parallel application?

SLURM

- Slurm workload manager
  - jobs scheduler for HPC clusters
  - used by all IT4I systems, LUMI, …
  - https://slurm.schedmd.com/
  - https://docs.it4i.cz/general/slurm-job-submission-and-execution/
  - https://docs.lumi-supercomputer.eu/runjobs/scheduled-jobs/slurm-quickstart/

SLURM

- `sbatch script.sh`
- `sbatch –-nodes 2 script.sh`

Slurm starts job:

- at submitted directory
- without any load module
  - do not forget to load modules

```
#!/usr/bin/bash
#SBATCH --job-name MyJobName
#SBATCH --account PROJECT-ID
#SBATCH --partition qcpu
#SBATCH --nodes 4
#SBATCH --ntasks-per-node 128
#SBATCH --time 12:00:00

ml purge
ml OpenMPI/4.1.4-GCC-11.3.0

srun hostname | sort | uniq -c
```

SLURM

Basic parameters

- `-N, --nodes`
- `-n, --ntasks`
  - total number of processes in the job
- `-c, --cpus-per-task`
  - number of cores per task
- `-t, --time`
  - Acceptable time formats include "minutes", "minutes:seconds", "hours:minutes:seconds", "days-hours", "days-hours:minutes" and "days-hours:minutes:seconds"
- `-D, --chdir`
- `-A, --account`
- `-J, --job-name`

SLURM

Interactive jobs:

- `salloc -A PROJECT-ID -p qcpu -N 4 --ntasks-per-node 128 -t 2:00:00`

Start the job:

- `srun –n 128 ./app`

Get info about queued jobs:

- `squeue –-me`
- `sacct`

Job canceling:

- `scancel JOBID`

Informations about nodes and partitions:

- `sinfo`

SLURM

Examples: 4_parallel_programming/jobs

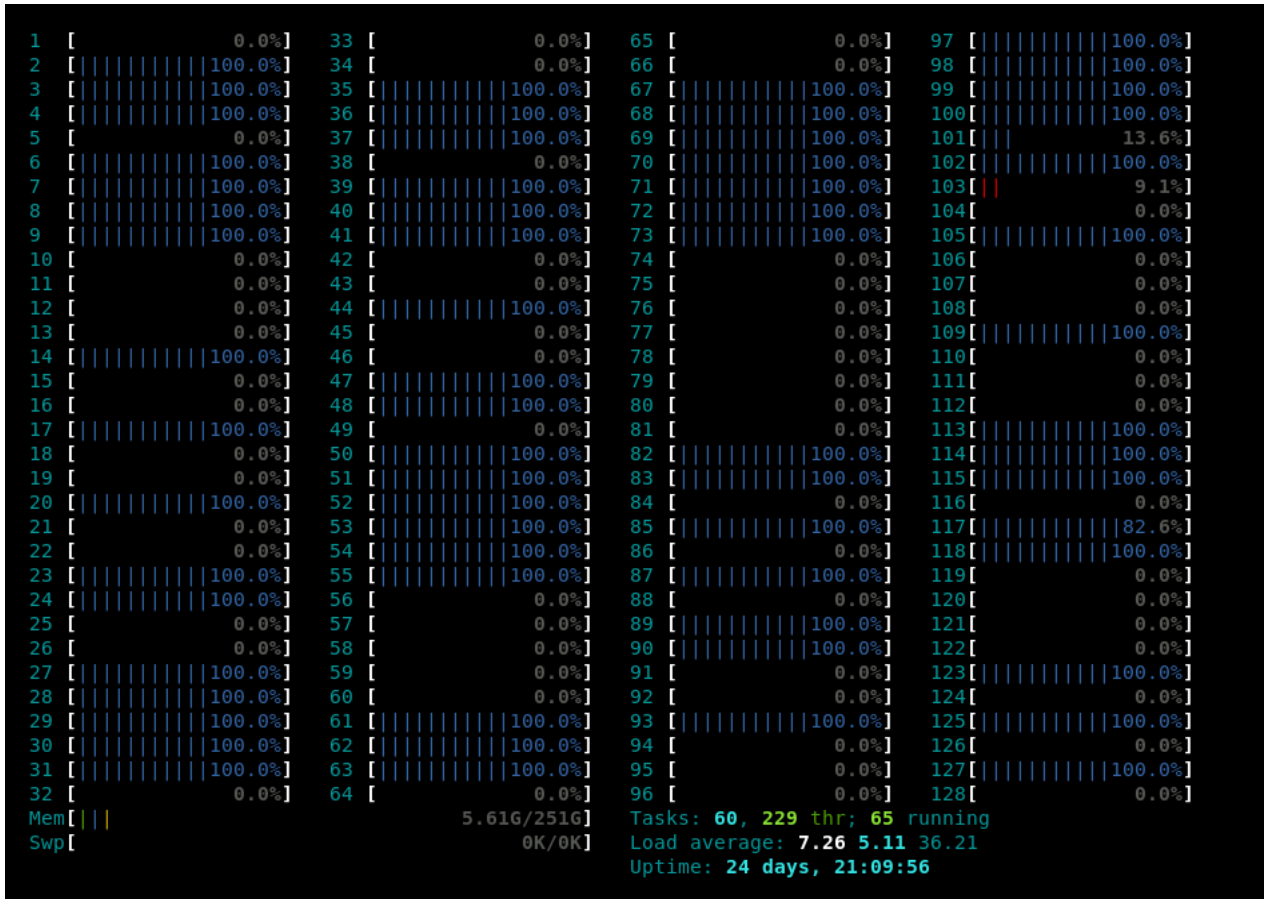
- hostaname.slurm: print hostname of cpu where the job is started
- parameters.slurm: list all parameters available in the job

- work.sequential.slurm: simple sequential job
- work.parallel.slurm: simple parallel job
- work.array.slurm: array jobs
- work.dependent.sh: jobs with dependency

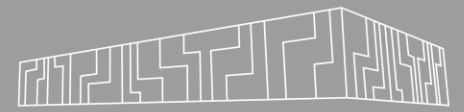
PARALLEL RUN

```
$ salloc -A DD-24-88 -p qcpu -N 1
$ export OMP_NUM_THREADS=64
$ srun -n 1 ./threaded
```

```
$ ssh cnXXX
$ htop -d2
```



PARALLEL RUN

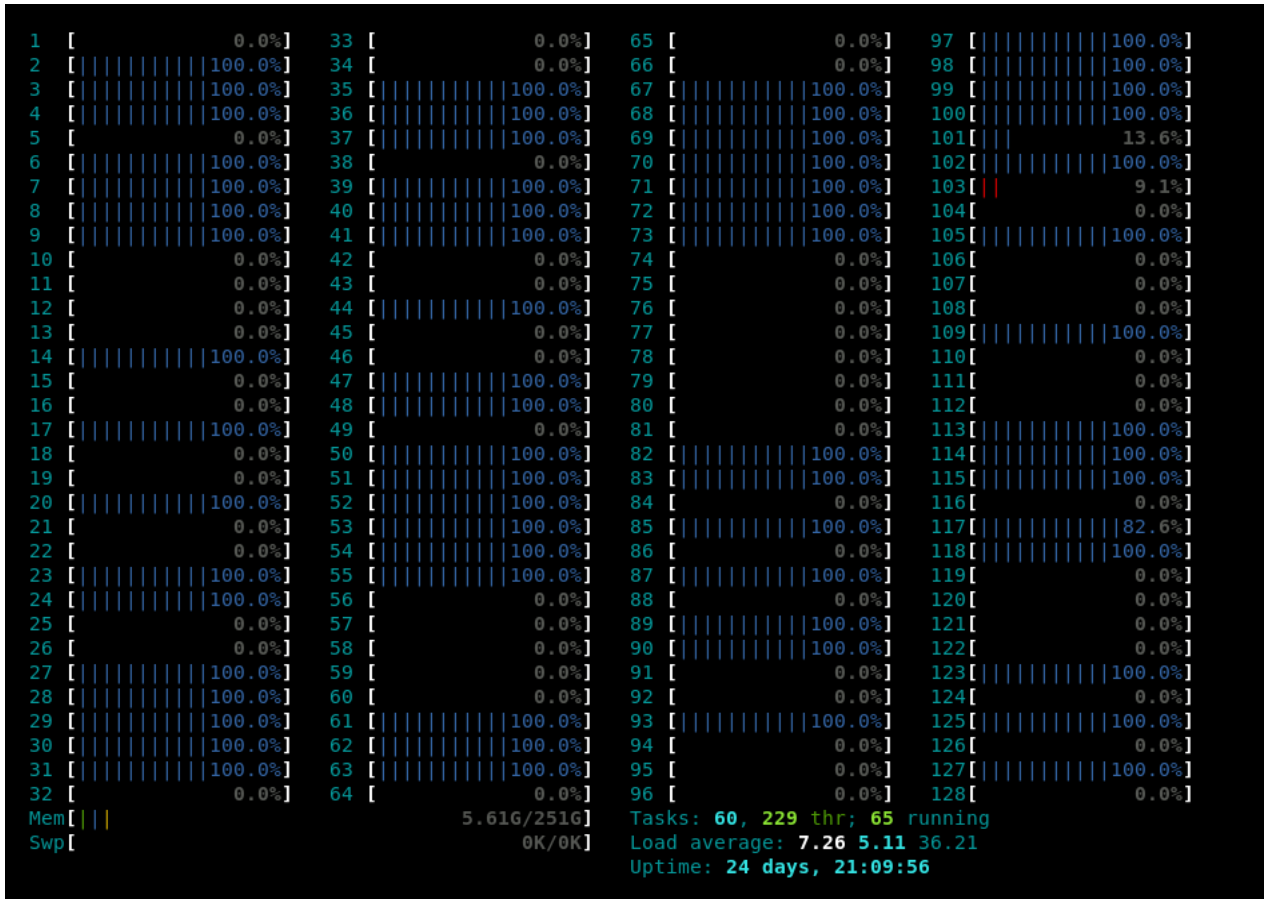


```
$ salloc -A DD-24-88 -p qcpu -N 1
$ export OMP_NUM_THREADS=64
$ srun -n 1 ./threaded
```

```
$ ssh cnXXX
$ htop -d2
```

How are threads pinned?

How will MPI be pinned?



PARALLEL RUN

```
$ salloc -A DD-24-88 -p qcpu -N 1
$ export OMP_NUM_THREADS=64
$ srun -n 1 ./threaded
```

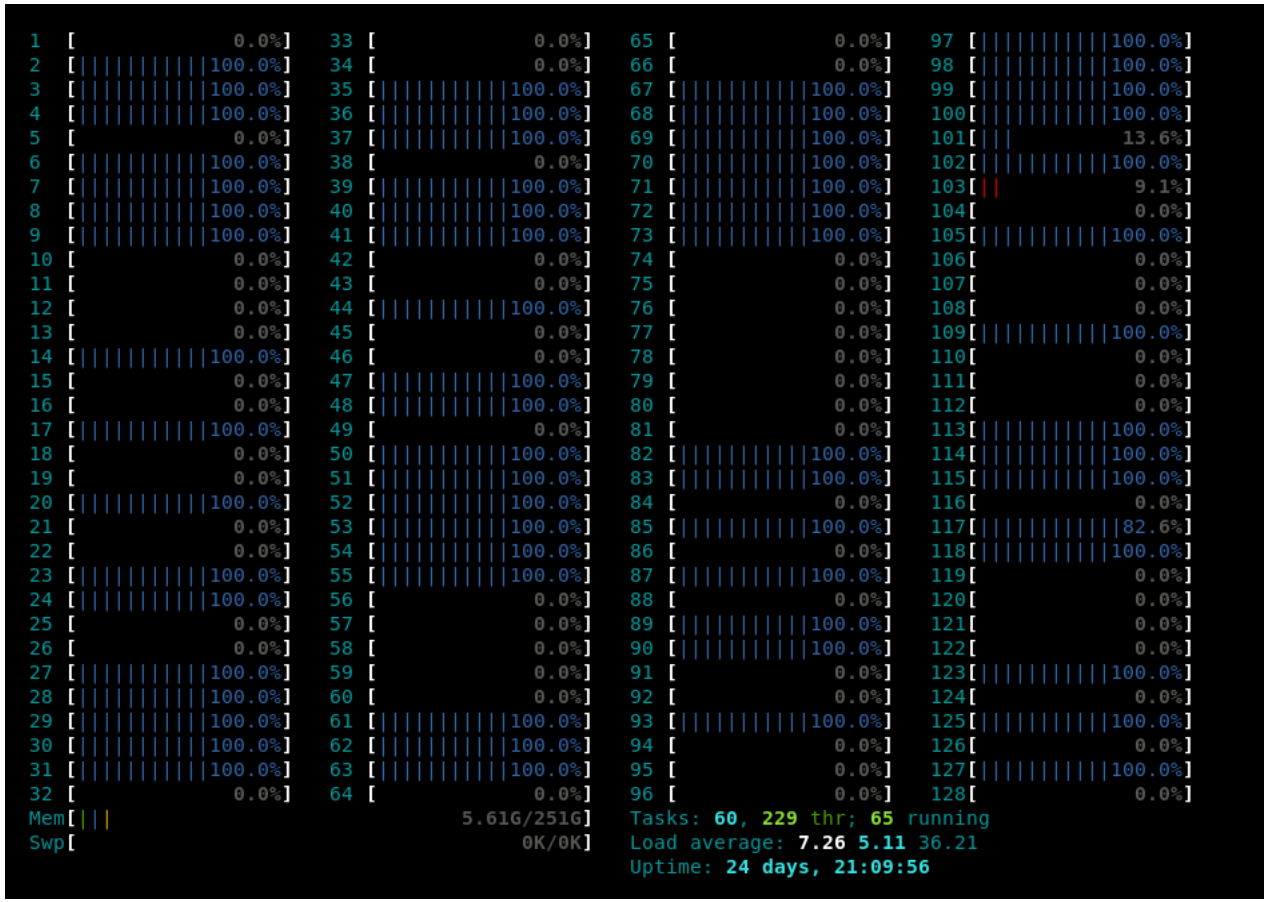
```
$ ssh cnXXX
$ htop -d2
```

How are threads pinned?

How will MPI be pinned?

Unfortunately:

- mapping significantly influence performance
- mapping is **highly non-portable**
  - different for OpenMPI, Intel, Slurm
  - dependent on a particular system



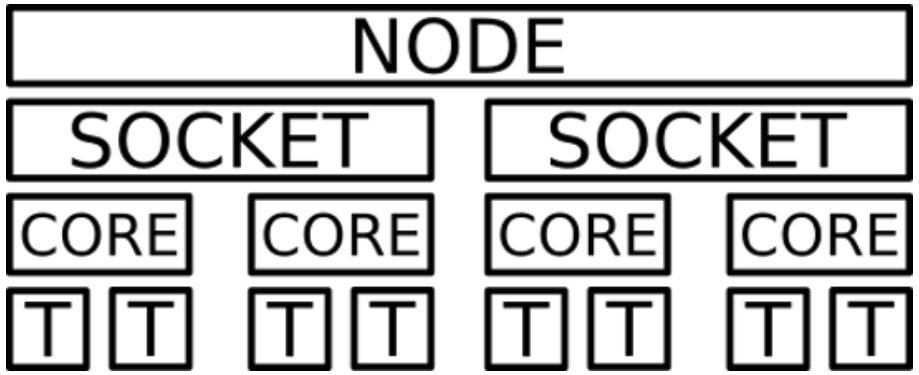
PARALLEL RUN

Environment variables

- OMP_NUM_THREADS
- OMP_PLACES=<threads, cores, sockets>
- OMP_PROC_BIND=<true, false, master, close, spread>

- Intel-MPI
  - KMP_AFFINITY
  - I_MPI_PIN_DOMAIN
- OpenMPI
  - --bind-to <hwthread, core, socket, numa, ...>
  - --map-by <hwthread, core, socket, numa, ...>
  - --report-bindings
- Slurm (srun)
  - --cpu-bind=<sockets,ldoms,cores>
  - -c, --cpus-per-task=<ncpus>

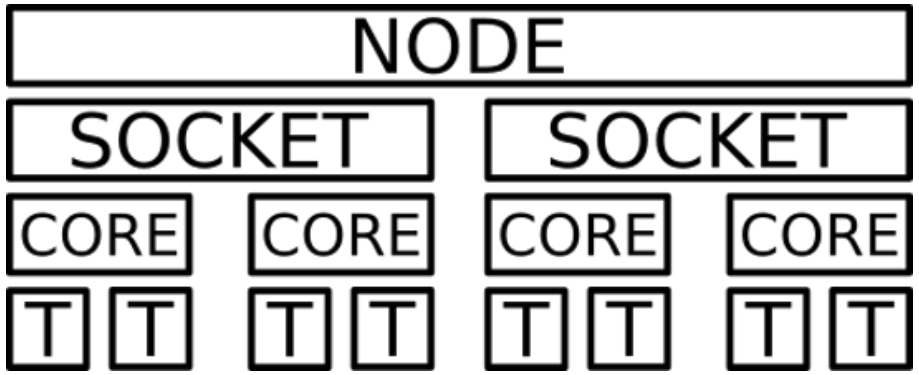
PARALLEL RUN - OPENMP



OMP_PROC_BIND=close

0 1 2 3 4 5 6 7

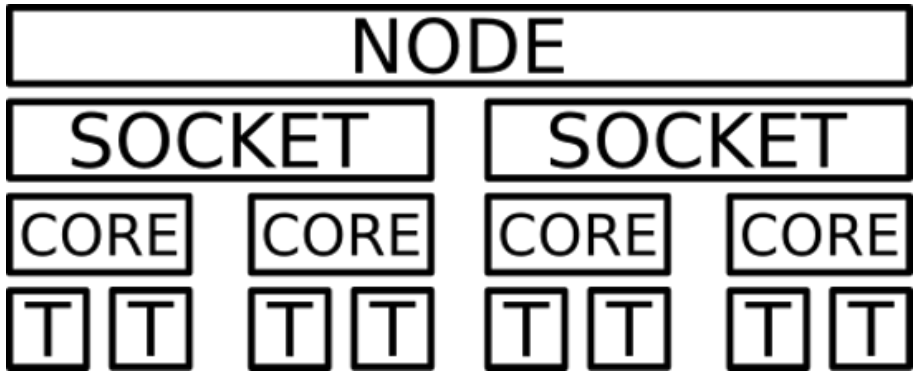
PARALLEL RUN - OPENMP



OMP_PROC_BIND=close 0 1 2 3 4 5 6 7

OMP_NUM_THREADS=2 OMP_PROC_BIND=spread 0 1

PARALLEL RUN - OPENMP



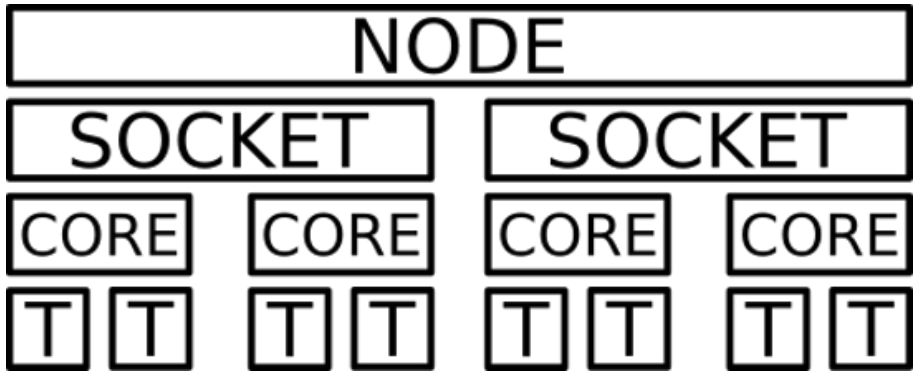
OMP_PROC_BIND=close	0	1	2	3	4	5	6	7
OMP_NUM_THREADS=2 OMP_PROC_BIND=spread	0				1			
OMP_NUM_THREADS=4 OMP_PROC_BIND=spread	0		1		2		3	

PARALLEL RUN – INTEL MPI

- KMP_AFFINITY=[<modifier>,...]<type>[,<permute>][,<offset>]
  - modifier:
    - verbose, warnings, respect
    - granularity= fine, **thread**, core, tile, die, node, group, and socket
  - type:
    - balanced, **compact**, disabled, explicit, none, scatter
  - permute
    - 0 – thread, 1 – core, 2 – socket
    - positive number (default 0)
  - offset
    - position where the first thread is assigned
    - positive number (default 0)

https://www.intel.com/content/www/us/en/develop/documentation/mpi-developer-reference-linux/top/environment-variable-reference/process-pinning/environment-variables-for-process-pinning.html

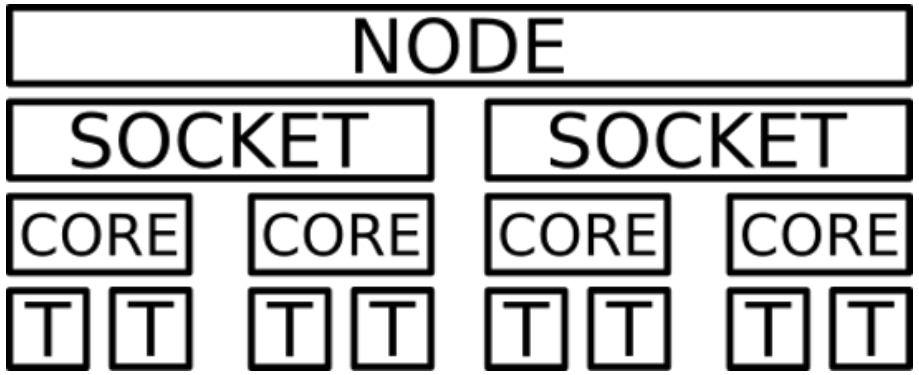
PARALLEL RUN – INTEL MPI



KMP_AFFINITY=granularity=thread,compact

0 1 2 3 4 5 6 7

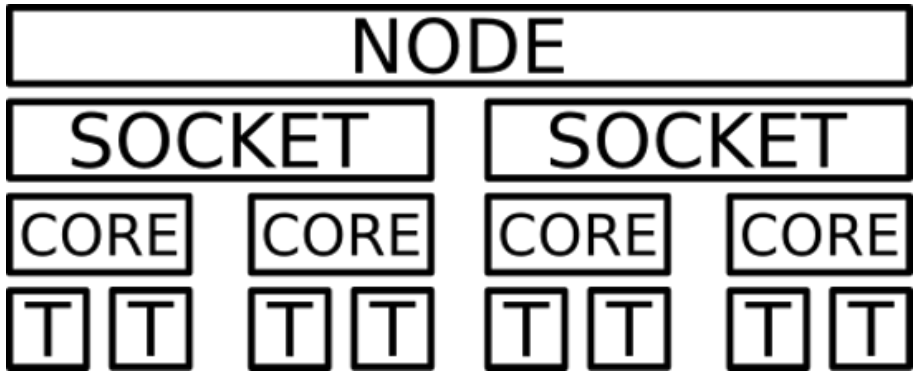
PARALLEL RUN – INTEL MPI



KMP_AFFINITY=granularity=thread,compact 0 1 2 3 4 5 6 7

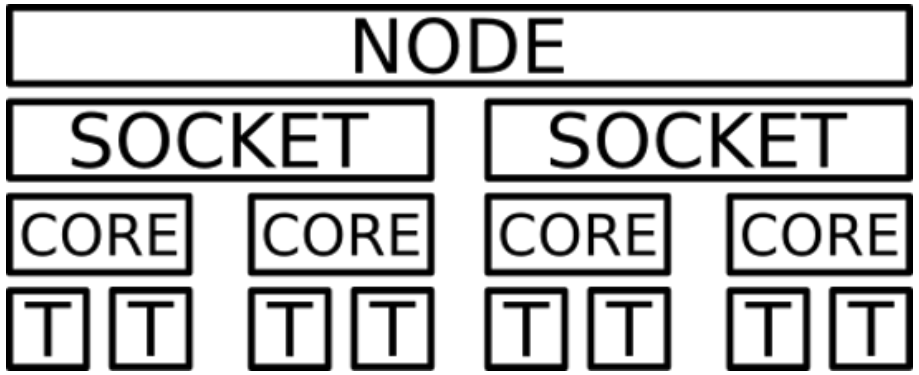
KMP_AFFINITY=granularity=thread,scatter 0

PARALLEL RUN – INTEL MPI



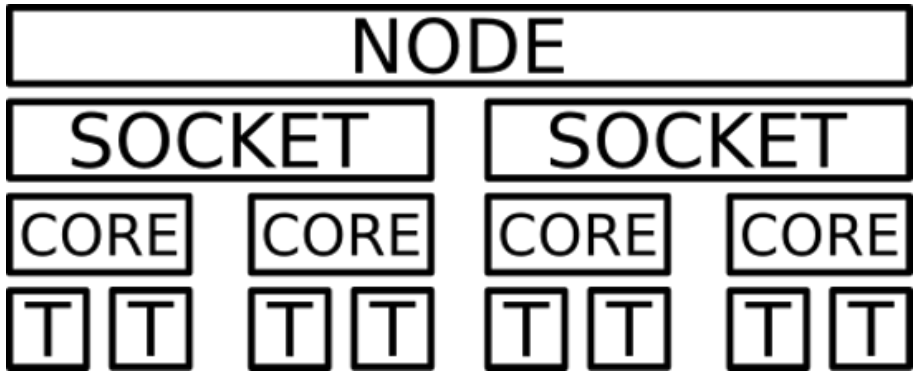
KMP_AFFINITY=granularity=thread,compact	0	1	2	3	4	5	6	7
KMP_AFFINITY=granularity=thread,scatter	0				1			

PARALLEL RUN – INTEL MPI



KMP_AFFINITY=granularity=thread,compact	0	1	2	3	4	5	6	7
KMP_AFFINITY=granularity=thread,scatter	0		2		1		3	

PARALLEL RUN – INTEL MPI

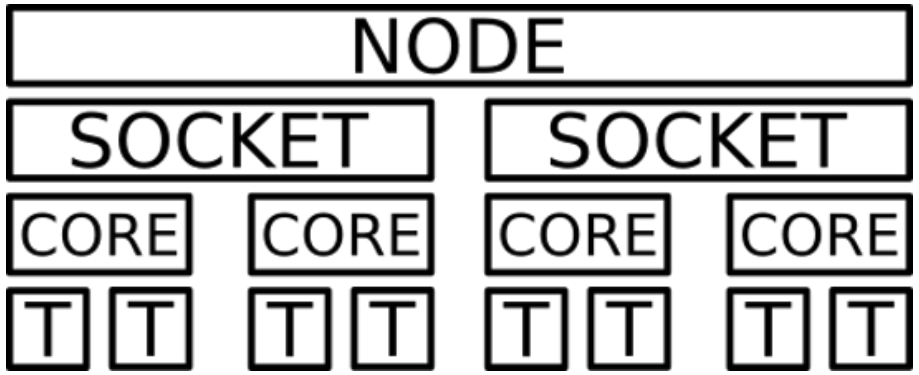


KMP_AFFINITY=granularity=thread,compact 0 1 2 3 4 5 6 7

KMP_AFFINITY=granularity=thread,scatter 0 4 2 6 1 5 3 7

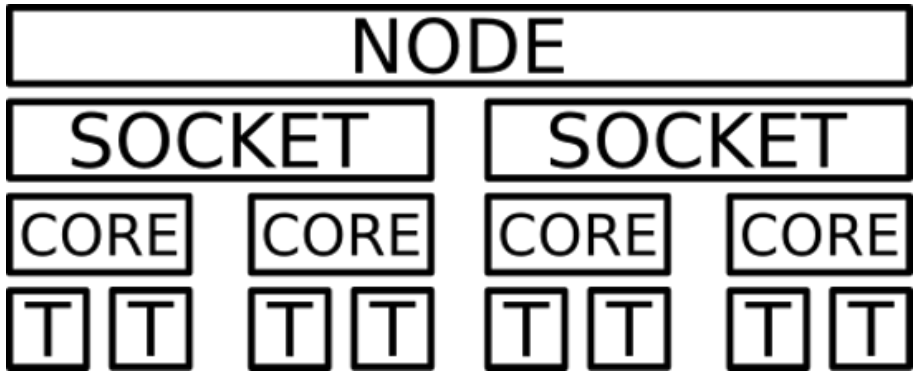
KMP_AFFINITY=granularity=thread,compact,0,5

PARALLEL RUN – INTEL MPI



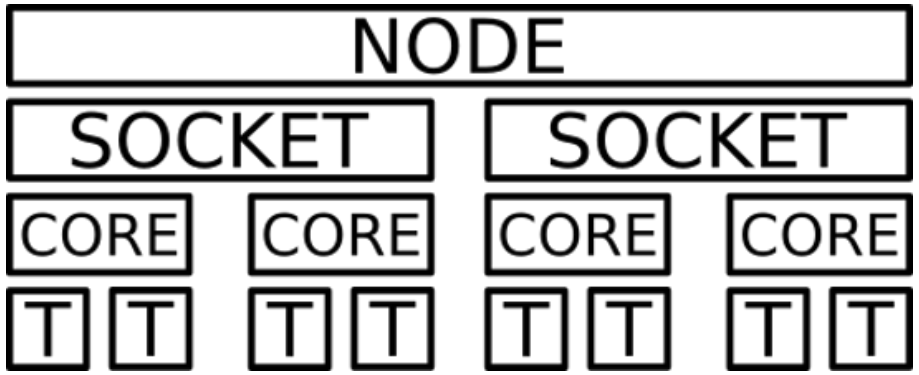
KMP_AFFINITY=granularity=thread,compact	0	1	2	3	4	5	6	7
KMP_AFFINITY=granularity=thread,scatter	0	4	2	6	1	5	3	7
KMP_AFFINITY=granularity=thread,compact,0,5						0		

PARALLEL RUN – INTEL MPI



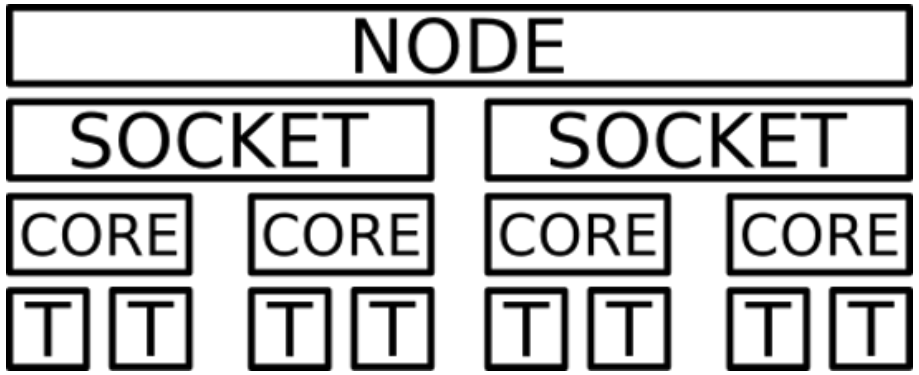
KMP_AFFINITY=granularity=thread,compact	0	1	2	3	4	5	6	7
KMP_AFFINITY=granularity=thread,scatter	0	4	2	6	1	5	3	7
KMP_AFFINITY=granularity=thread,compact,0,5						0	1	

PARALLEL RUN – INTEL MPI



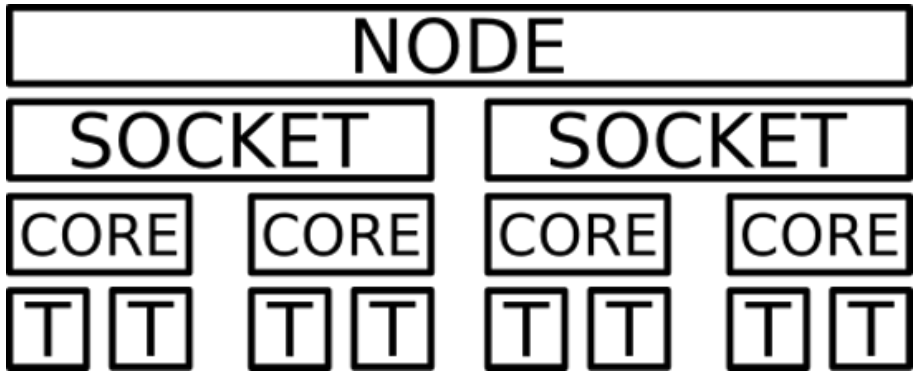
KMP_AFFINITY=granularity=thread,compact	0	1	2	3	4	5	6	7
KMP_AFFINITY=granularity=thread,scatter	0	4	2	6	1	5	3	7
KMP_AFFINITY=granularity=thread,compact,0,5	3	4	5	6	7	0	1	2

PARALLEL RUN – INTEL MPI



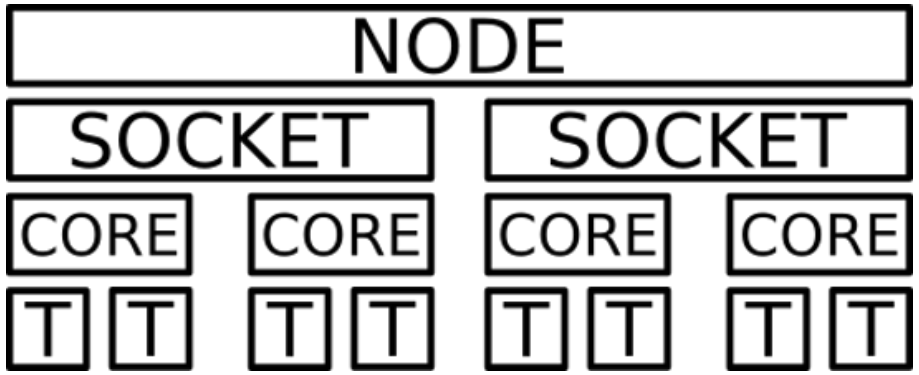
KMP_AFFINITY=granularity=thread,compact	0	1	2	3	4	5	6	7
KMP_AFFINITY=granularity=thread,scatter	0	4	2	6	1	5	3	7
KMP_AFFINITY=granularity=thread,compact,0,5	3	4	5	6	7	0	1	2
KMP_AFFINITY=granularity=thread,compact,1,0								

PARALLEL RUN – INTEL MPI



KMP_AFFINITY=granularity=thread,compact	0	1	2	3	4	5	6	7
KMP_AFFINITY=granularity=thread,scatter	0	4	2	6	1	5	3	7
KMP_AFFINITY=granularity=thread,compact,0,5	3	4	5	6	7	0	1	2
KMP_AFFINITY=granularity=thread,compact,1,0	0		1		2		3	

PARALLEL RUN – INTEL MPI

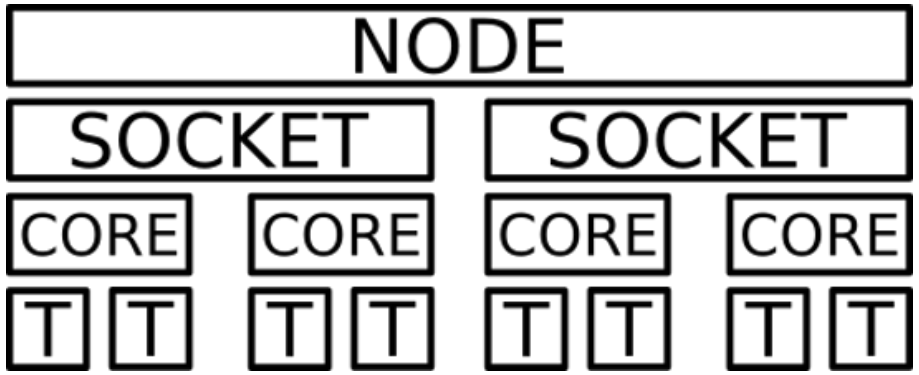


KMP_AFFINITY=granularity=thread,compact	0	1	2	3	4	5	6	7
KMP_AFFINITY=granularity=thread,scatter	0	4	2	6	1	5	3	7
KMP_AFFINITY=granularity=thread,compact,0,5	3	4	5	6	7	0	1	2
KMP_AFFINITY=granularity=thread,compact,1,0	0	4	1	5	2	6	3	7

PARALLEL RUN – INTEL MPI

- I_MPI_PIN_DOMAIN=[shape]
  - <size>[:<layout>]
    - number of logical processors in each domain with a layout (platform, compact, scatter)
  - core
  - socket
  - numa
  - cache

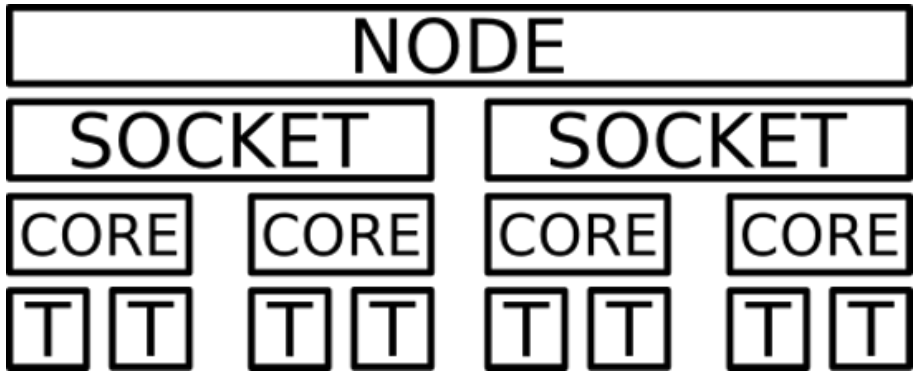
PARALLEL RUN – INTEL MPI



I_MPI_PIN_DOMAIN=4

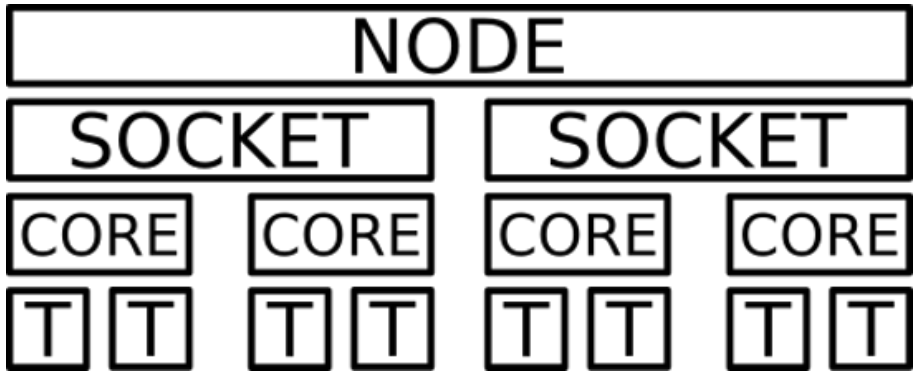
0 0 0 0 1 1 1 1

PARALLEL RUN – INTEL MPI



I_MPI_PIN_DOMAIN=4	0	0	0	0	1	1	1	1
I_MPI_PIN_DOMAIN=2	0	0	1	1	2	2	3	3

PARALLEL RUN – INTEL MPI



I_MPI_PIN_DOMAIN=4 0 0 0 0 1 1 1 1

I_MPI_PIN_DOMAIN=2 0 0 1 1 2 2 3 3

I_MPI_PIN_DOMAIN=$OMP_NUM_THREADS

I_MPI_PIN_DOMAIN=socket

I_MPI_PIN_DOMAIN=cache3

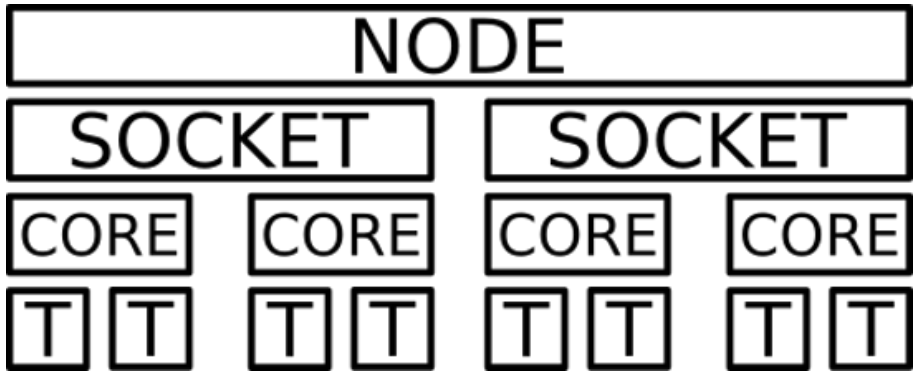
I_MPI_PIN_RESPECT_HCA=0 pinning does not respect host channel adapter

PARALLEL RUN – OPEN MPI

- --bind-to <hwthread, **core**, l3cache, numa, socket, ppr, …>
  - bind to the processors associated with hardware component
- --map-by <hwthread, core, l3cache, numa, **socket**, …>
  - map across the specified hardware component
- --report-bindings

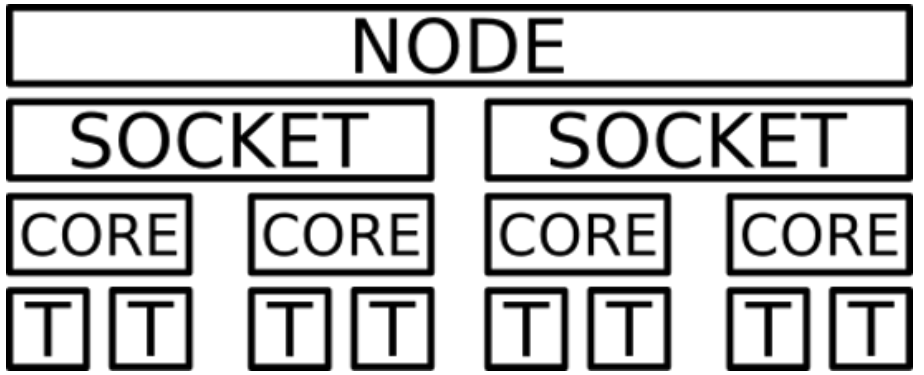
https://www.open-mpi.org/doc/v3.0/man1/mpirun.1.php

PARALLEL RUN – OPEN MPI




```
export OMP_PROC_BIND=close
export OMP_NUM_THREADS=2
mpirun -n 4 --map-by core --bind-to core ./app
```

PARALLEL RUN – OPEN MPI



export OMP_PROC_BIND=close

export OMP_NUM_THREADS=2

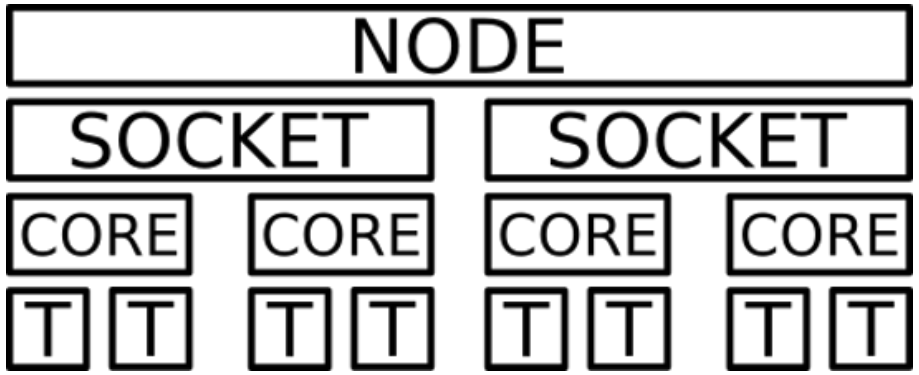
mpirun -n 4 --map-by core --bind-to core ./app

```
0   0   1   1   2   2   3   3
```

mpirun -n 4 --map-by socket --bind-to socket ./app

```
0|2     0|2     1|3     1|3
```

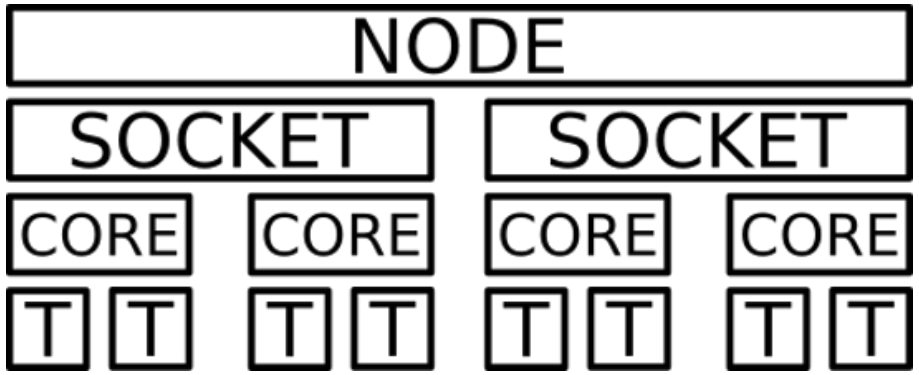
PARALLEL RUN – OPEN MPI




```
export OMP_PROC_BIND=close
export OMP_NUM_THREADS=2
mpirun -n 4 --map-by core --bind-to core ./app
mpirun -n 4 --map-by socket --bind-to socket ./app
mpirun -n 4 --map-by socket --bind-to core ./app
```

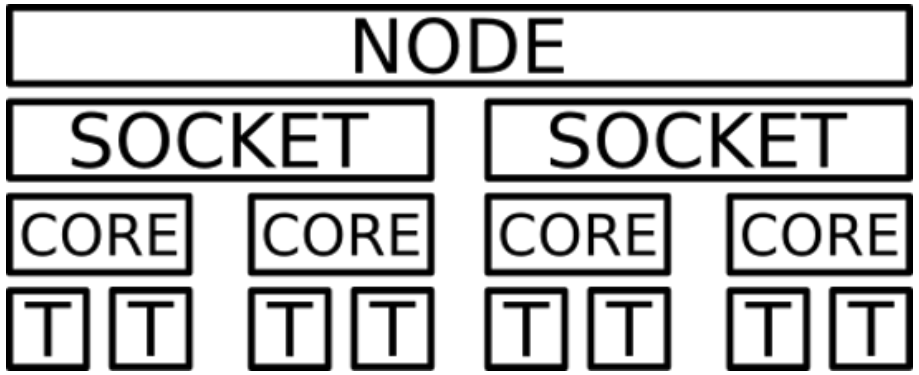
```
0   0   1   1   2   2   3   3
0|2     0|2     1|3     1|3
0|2 -   -   -   1|3 -   -   -
```

PARALLEL RUN – OPEN MPI



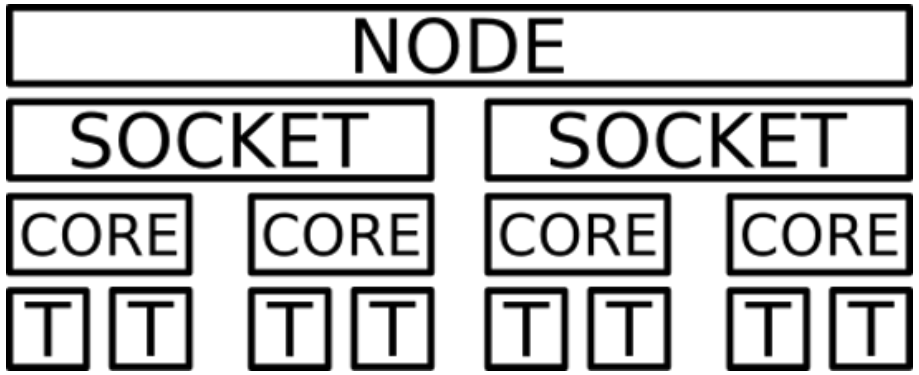

```
export OMP_PROC_BIND=close
export OMP_NUM_THREADS=2
mpirun -n 4 --map-by core --bind-to core ./app          0    0    1    1    2    2    3    3
mpirun -n 4 --map-by socket --bind-to socket ./app       0|2       0|2       1|3       1|3
mpirun -n 4 --map-by socket --bind-to core ./app         0|2  -    -    -     1|3  -    -    -
mpirun -n 4 --map-by thread --bind-to socket ./app     0|1|2|3 0|1|2|3   ---       ---
```

PARALLEL RUN – OPEN MPI




```
export OMP_PROC_BIND=close
export OMP_NUM_THREADS=2
mpirun -n 4 --map-by core --bind-to core ./app          0    0    1    1    2    2    3    3
mpirun -n 4 --map-by socket --bind-to socket ./app       0|2       0|2       1|3       1|3
mpirun -n 4 --map-by socket --bind-to core ./app         0|2 -   -    -       1|3 -   -    -
mpirun -n 4 --map-by thread --bind-to socket ./app    0|1|2|3 0|1|2|3   ---       ---
mpirun -n 4 --map-by numa --bind-to numa ./app           0|2       0|2       1|3       1|3
```

PARALLEL RUN – OPEN MPI

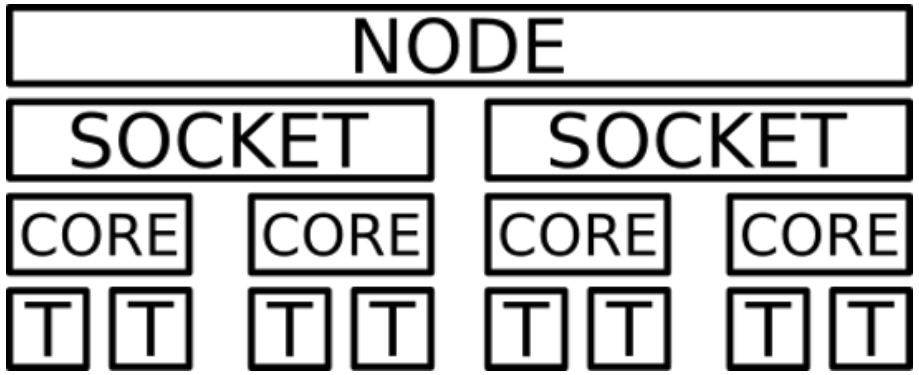



```
export OMP_PROC_BIND=close
export OMP_NUM_THREADS=2
mpirun -n 4 --map-by core --bind-to core ./app          0    0    1    1    2    2    3    3
mpirun -n 4 --map-by socket --bind-to socket ./app       0|2       0|2       1|3       1|3
mpirun -n 4 --map-by socket --bind-to core ./app         0|2 -   -    -     1|3 -   -    -
mpirun -n 4 --map-by thread --bind-to socket ./app    0|1|2|3 0|1|2|3   ---       ---
mpirun -n 4 --map-by numa --bind-to numa ./app           0|2       0|2       1|3       1|3
```

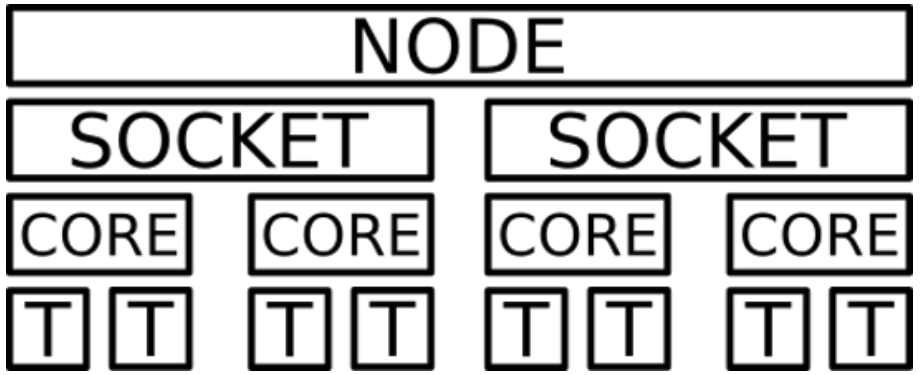
OpenMPI defaults
--bin-to core (when the number of processes is <= 2)
--bind-to socket (when the number of processes is > 2)

PARALLEL RUN – SLURM

```
export OMP_PROC_BIND=close
export OMP_NUM_THREADS=1
srun -n 2 -c 2 ./app
```



PARALLEL RUN – SLURM




```
export OMP_PROC_BIND=close
export OMP_NUM_THREADS=1
srun -n 2 -c 2 ./app
srun -n 4 -c 1 ./app
```

```
0       1
0   1   2   3
```

PARALLEL RUN

What is the optimal setting?

PARALLEL RUN

What is the optimal setting?

- hardware configuration
  - number of NUMA domains
  - caches, memory channels,...
- application features
  - OpenMP only
  - pure MPI
  - hybrid parallelization

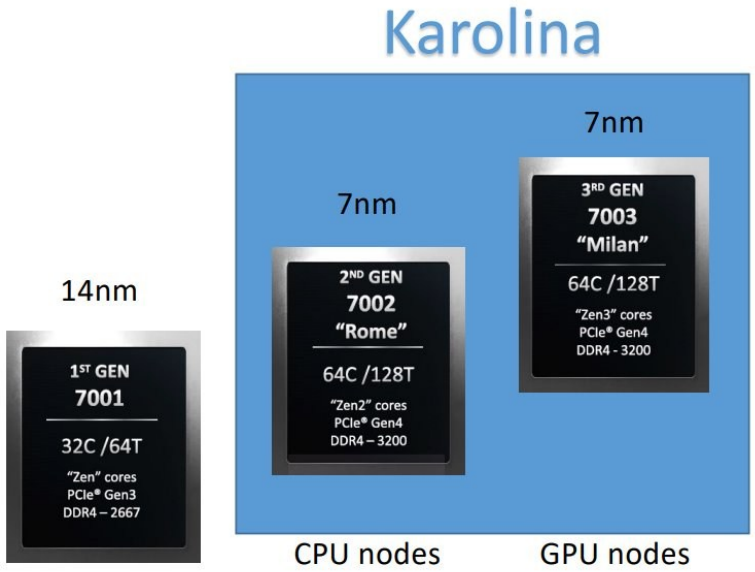
PARALLEL APPLICATIONS

Pure MPI application:

- salloc -N 1 –n 128 ...
- srun –n 128 ./app

MPI+OpenMP application:

- salloc -N 1 –n 32 ...
- OMP_NUM_THREADS=4 srun –n 32 ./app

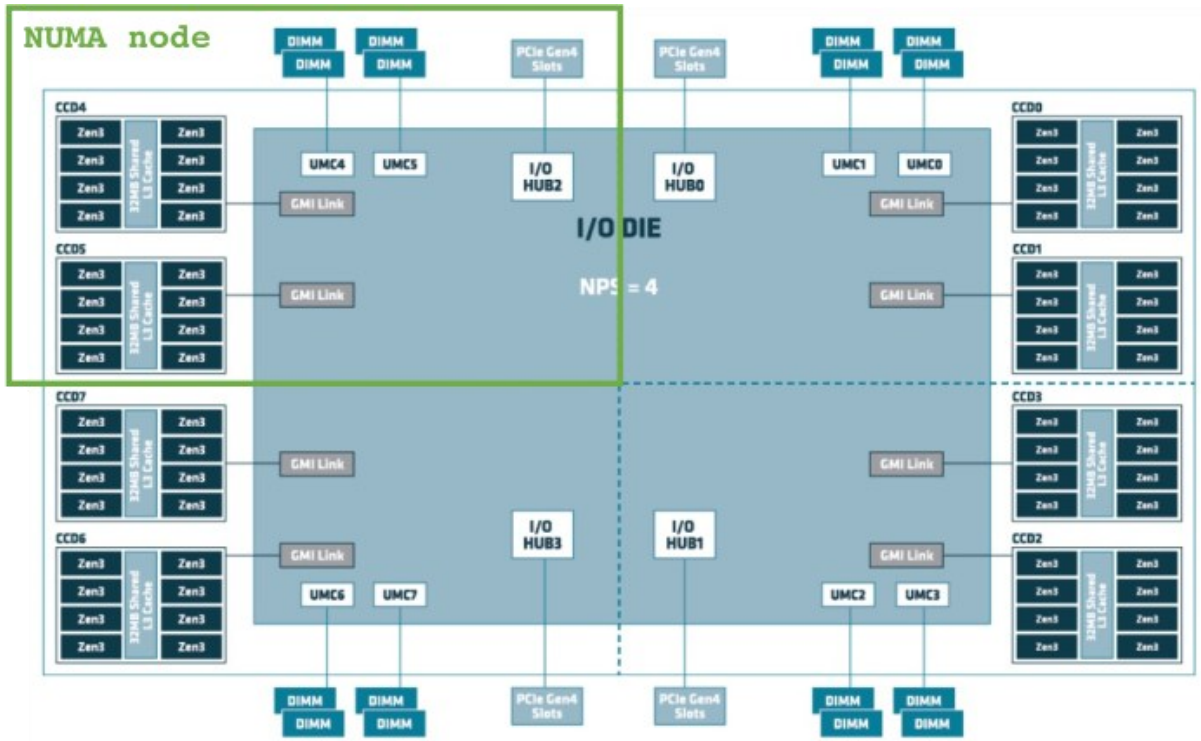


CATEGORY	EPYC 7002 (Rome)	EPYC 7003 (Milan)
Socket	SP3	SP3
Core / Process	Zen2 / 7nm	Zen3 / 7nm
Max Core Count / Threads	64 / 128	64 / 128
L3 Cache Size	256 MB	256 MB
CCX Arch	4 Cores + 16MB	8 Cores + 32MB
Memory	8 Ch DDR4-3200, NVDIMM-N	8 Ch DDR4-3200, NVDIMM-N
PCIe Tech & Lane Count	PCIe Gen4, 128L/Socket	PCIe Gen4, 128L/Socket
Security	SME, SEV	SME, SEV, SNP
Chipset	NA	NA
Power	120W - 280W	120W - 280W

Node architecture

```
numactl -H
| node 0 cpus:   0 -  15
| node 1 cpus:  16 -  31
| node 2 cpus:  32 -  47
| node 3 cpus:  48 -  63
| node 4 cpus:  64 -  79
| node 5 cpus:  80 -  95
| node 6 cpus:  96 - 111
| node 7 cpus: 112 - 127
| node 0-7 size: 128GB
```

	0	1	2	3	4	5	6	7
0	10	12	12	12	32	32	32	32
1	12	10	12	12	32	32	32	32
2	12	12	10	12	32	32	32	32
3	12	12	12	10	32	32	32	32
4	32	32	32	32	10	12	12	12
5	32	32	32	32	12	10	12	12
6	32	32	32	32	12	12	10	12
7	32	32	32	32	12	12	12	10



PARALLEL APPLICATIONS

I have a simple application:

- Karolina supercomputer at IT4I
- compile with OpenMPI: `mpic++ -fopenmp -O3 -march=native app.cpp -o app`
- test with different number of MPI processors up to 128

- salloc -p qcpu_exp -N 1
- export OMP_NUM_THREADS=1

- srun –n 8 ./app
- srun –n 16 ./app
- srun –n 32 ./app
- srun –n 64 ./app
- srun –n 128 ./app

PARALLEL APPLICATIONS

I have a simple application:

- Karolina supercomputer at IT4I
- compile with OpenMPI: `mpic++ -fopenmp -O3 -march=native app.cpp -o app`
- test with different number of MPI processors up to 128

- salloc -p qcpu_exp -N 1
- export OMP_NUM_THREADS=1

- srun –n 8 ./app 39.66s
- srun –n 16 ./app 17.46s
- srun –n 32 ./app 11.87s
- srun –n 64 ./app 7.31s
- srun –n 128 ./app 5.56s

Is 128 the best possible settings?

What about mapping and pinning?

PARALLEL APPLICATIONS

I have a simple application:

- Karolina supercomputer at IT4I
- compile with OpenMPI: `mpic++ -fopenmp -O3 -march=native app.cpp -o app`
- test with different number of MPI processors up to 128

- salloc -p qcpu_exp -N 1
- export OMP_NUM_THREADS=1

srun –n 8 ./app 39.66s	srun –n 8 –c 16 ./app 7.11s
srun –n 16 ./app 17.46s	srun –n 16 –c 8 ./app 5.21s
srun –n 32 ./app 11.87s	srun –n 32 –c 4 ./app 5.08s
srun –n 64 ./app 7.31s	srun –n 64 –c 2 ./app 5.31s
srun –n 128 ./app 5.56s	srun –n 128 –c 1 ./app 5.56s

PARALLEL APPLICATIONS

I have a simple application:

- Karolina supercomputer at IT4I
- compile with OpenMPI: `mpic++ -fopenmp -O3 -march=native app.cpp -o app`
- test with different number of MPI processors up to 128

- salloc -p qcpu_exp -N 1
- export OMP_NUM_THREADS=1

srun –n 8 ./app	39.66s	srun –n 8 –c 16 ./app	7.11s
srun –n 16 ./app	17.46s	srun –n 16 –c 8 ./app	5.21s
srun –n 32 ./app	11.87s	srun –n 32 –c 4 ./app	**5.08s (91% of the previous best)**
srun –n 64 ./app	7.31s	srun –n 64 –c 2 ./app	5.31s
srun –n 128 ./app	**5.56s**	srun –n 128 –c 1 ./app	5.56s

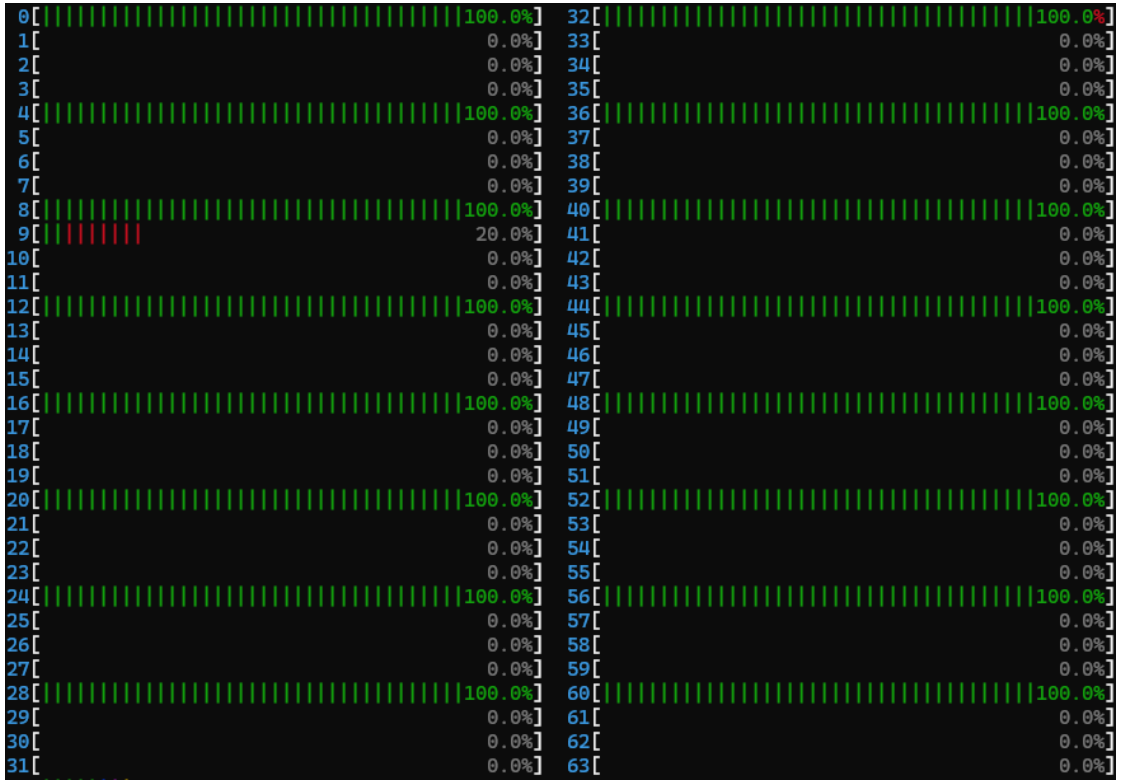
PARALLEL APPLICATIONS

I have a simple application:

- Karolina supercomputer at IT4I
- compile with OpenMPI: `mpic++ -fopenmp -O3 -march=native app.cpp -o app`
- test with different number of MPI processors up to 128

- salloc -p qcpu_exp -N 1
- export OMP_NUM_THREADS=**32**
- export OMP_PROC_BIND=**spread**

- srun -n 1 ./app 5.08s



RECOMMENDATION

Memory bound application

- Number of MPI processes / thread equal to memory channels
- Correct pinning to NUMA domains

Compute bound application

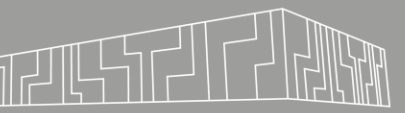
- As many MPI processes / threads as possible
- pinning to avoid migration

Your application

- Test performance for different number of cores per node:
  - 16, 32, 64, 128 cores per node

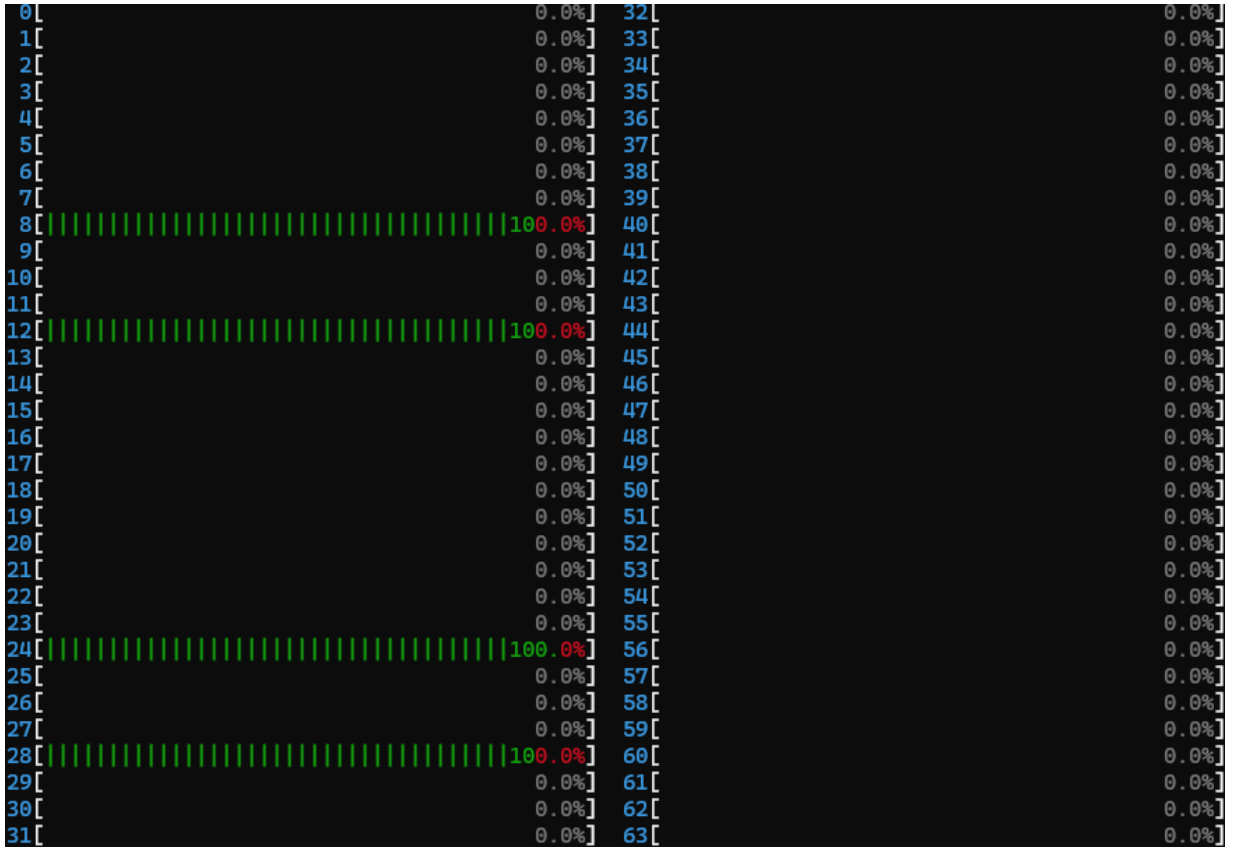
- Test different mapping / pinning options
  - close, spread

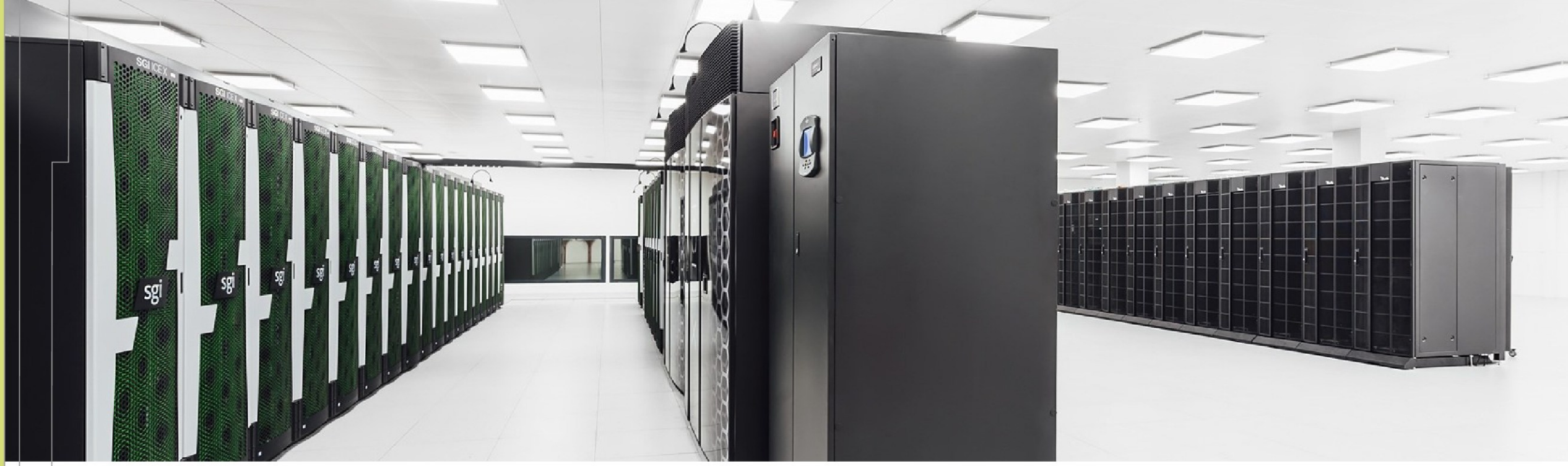
PARALLEL APPLICATIONS



More advanced binding

srun -n 4 --cpu-bind=mask_cpu: 0x100,0x1000,0x1000000,0x10000000 ./app





Ondřej Meca
ondrej.meca@vsb.cz

IT4Innovations National Supercomputing Center
VSB – Technical University of Ostrava
Studentská 6231/1B
708 00 Ostrava-Poruba, Czech Republic
www.it4i.cz

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS