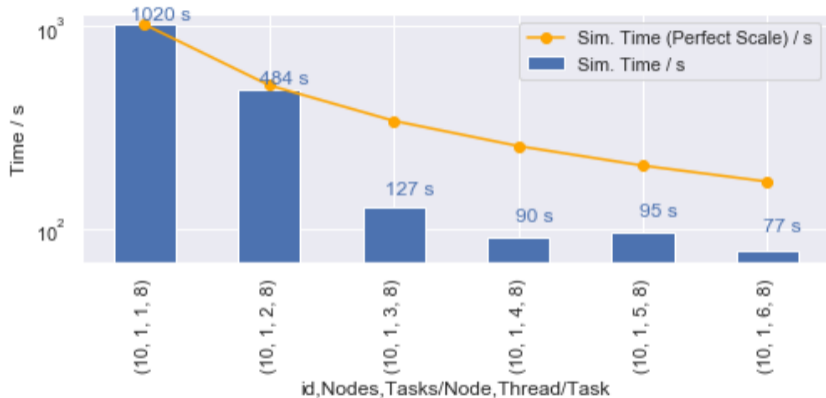


*p*⁶

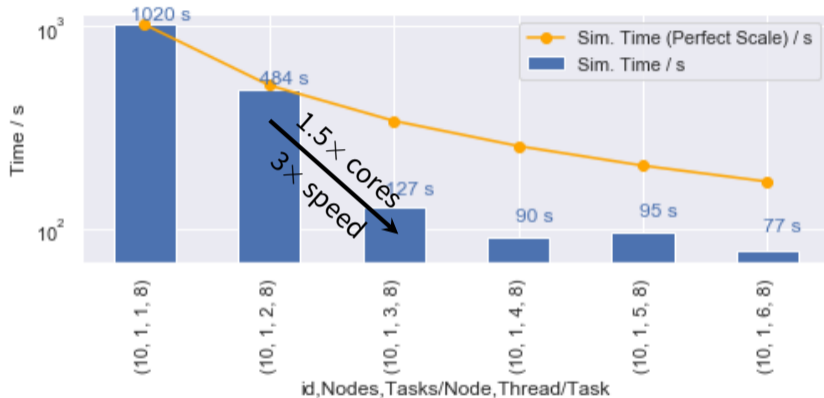
PROPER PINNING PREVENTS PRETTY POOR PERFORMANCE

May '23 | T. Hater | JSC

Superlinear Speed-Up?



Superlinear Speed-Up?



Superlinear Speed-Up?

No, just a bad baseline...

- Default process placement switched between two cases.

Superlinear Speed-Up?

No, just a bad baseline...

- Default process placement switched between two cases.
- Second configuration is better for this benchmark.

STREAM benchmark

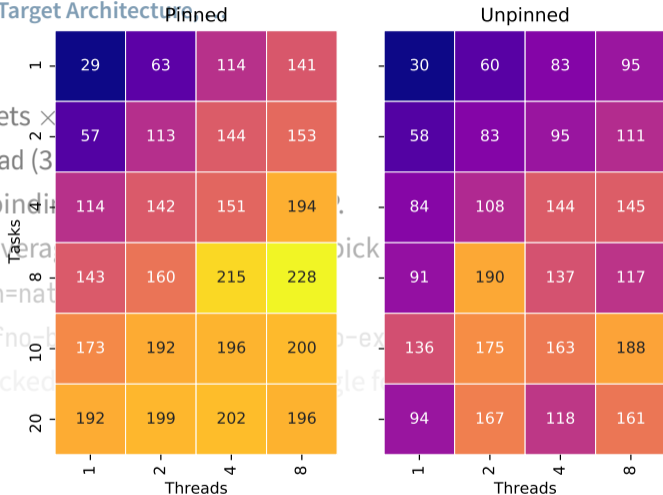
Heavily Optimised for Target Architecture, ...

- Target: 2 sockets \times 10 cores \times 8-way SMT
- 1GiB, only triad (3 double per element).
- De-activated bindings by MPI and OpenMP.
- 10 runs each averaged over 5 repetitions, pick top result.
- `-Ofast -march=native -mtune=native`
- `-std=c++17 -fno-builtin -fno-rtti -fno-exceptions -fopenmp`
- Cache line blocked and aligned, SIMD, single fork/join, first touch aware, RMW optimised.

STREAM benchmark

Heavily Optimised for Target Architecture

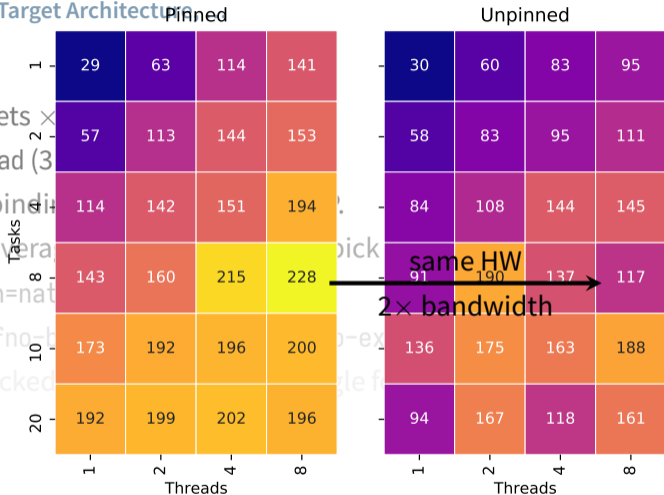
- Target: 2 sockets
- 1GiB, only triad (3)
- De-activated binding
- 10 runs each averaged
- -Ofast -march=native
- -std=c++17 -fno-bulk-memory-align
- Cache line blocked



STREAM benchmark

Heavily Optimised for Target Architecture

- Target: 2 sockets
- 1GiB, only triad (3)
- De-activated bindi
- 10 runs each avera
- Ofast -march=nat
- std=c++17 -fno-t
- Cache line blocked

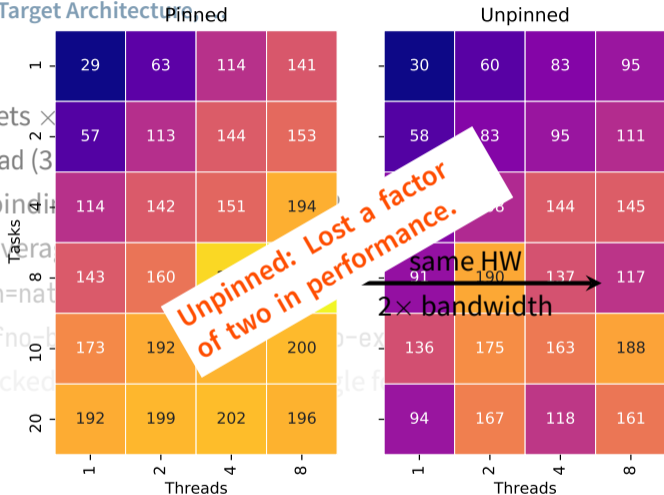


re, RMW optimised.

STREAM benchmark

Heavily Optimised for Target Architecture

- Target: 2 sockets
- 1GiB, only triad (3 elements)
- De-activated binding
- 10 runs each averaged
- -Ofast -march=native
- -std=c++17 -fno-builtin-memset
- Cache line blocked



What is Pinning?

Also: Binding, Affinity, ...

- Force a process or thread to execute only on a given set of cores.

What is Pinning?

Also: Binding, Affinity, ...

- Force a process or thread to execute only on a given set of cores.
- Increases performance predictability and absolute performance.

What is Pinning?

Also: Binding, Affinity, ...

- Force a process or thread to execute only on a given set of cores.
- Increases performance predictability and absolute performance.
- Enforced by the OS, driven by user space tools.

What is Pinning?

Also: Binding, Affinity, ...

- Force a process or thread to execute only on a given set of cores.
- Increases performance predictability and absolute performance.
- Enforced by the OS, driven by user space tools.
- In HPC this is (partially!) handled by the scheduler (SLURM) or MPI.

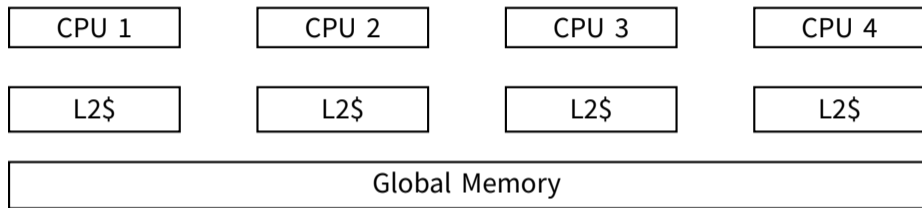
What is Pinning?

Also: Binding, Affinity, ...

- Force a process or thread to execute only on a given set of cores.
- Increases performance predictability and absolute performance.
- Enforced by the OS, driven by user space tools.
- In HPC this is (partially!) handled by the scheduler (SLURM) or MPI.
- But you can (should?) take control.

Why Pinning?

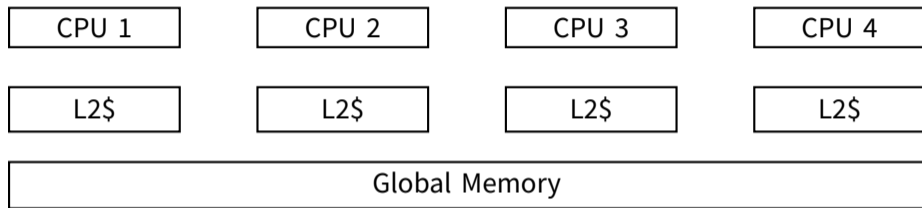
A Cartoon CPU



- Many cores, each with its own memory hierachy.

Why Pinning?

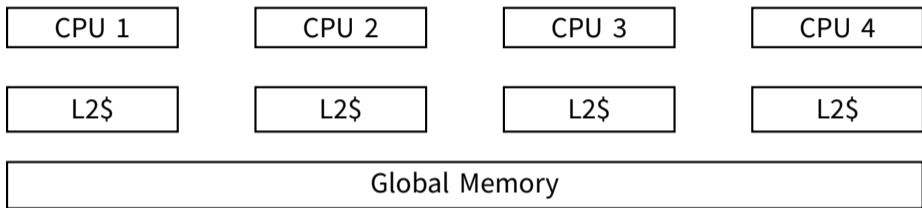
A Cartoon CPU



- Many cores, each with its own memory hierachy.
- Shared global memory, but...

Why Pinning?

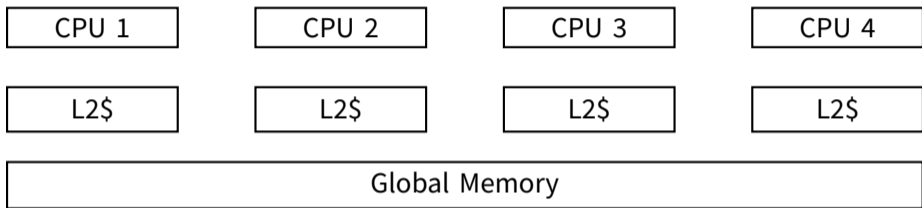
A Cartoon CPU



- Many cores, each with its own memory hierachy.
- Shared global memory, but...
- ...*affinity* to memory partitions.

Why Pinning?

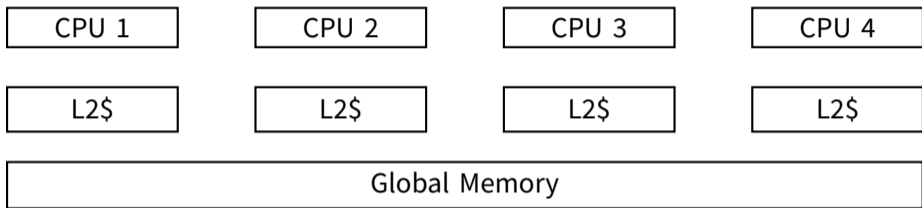
A Cartoon CPU



- Many cores, each with its own memory hierachy.
- Shared global memory, but...
- ...*affinity* to memory partitions.
- OS manages allocation,...

Why Pinning?

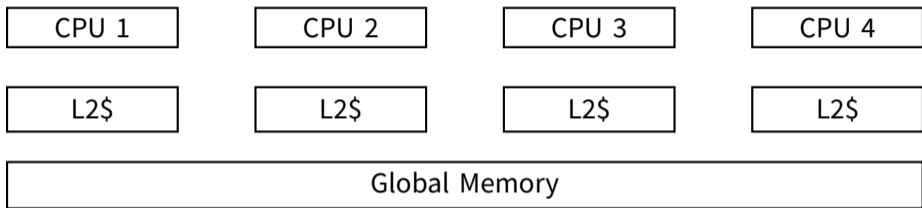
A Cartoon CPU



- Many cores, each with its own memory hierachy.
- Shared global memory, but...
- ...*affinity* to memory partitions.
- OS manages allocation,...
- ...task placement, and...

Why Pinning?

A Cartoon CPU



- Many cores, each with its own memory hierachy.
- Shared global memory, but...
- ...*affinity* to memory partitions.
- OS manages allocation,...
- ...task placement, and...
- ...swaps tasks in and out.

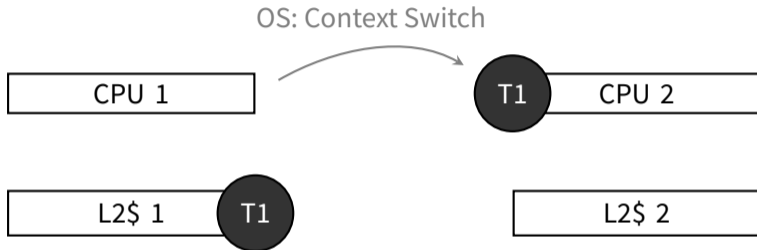
Why Pinning?

Scenario 1: Task Migration



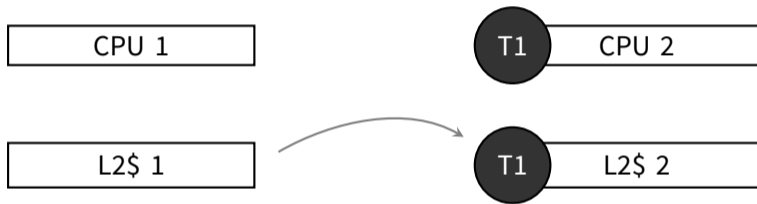
Why Pinning?

Scenario 1: Task Migration



Why Pinning?

Scenario 1: Task Migration



Why Pinning?

Scenario 1: Task Migration



Important

Swapping tasks in and out is basically free, but task *migration* leads to data migration. Granularity is a *cache line* (often 128 B); be aware of *false sharing*.

Why Pinning?

Scenario 2: NUMA

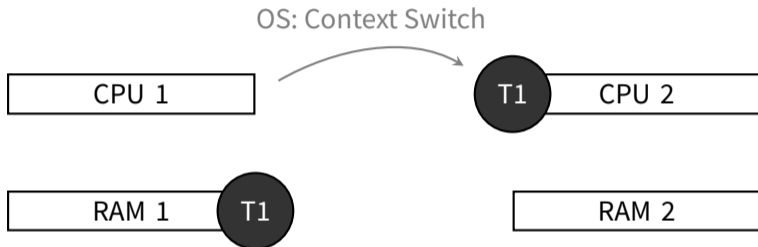
NUMA: Non-Uniform Memory Access, ie memory performance depends on relative location.



Why Pinning?

Scenario 2: NUMA

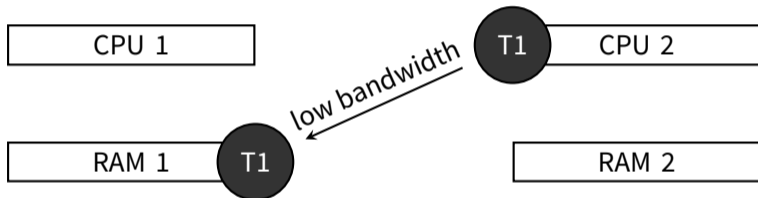
NUMA: Non-Uniform Memory Access, ie memory performance depends on relative location.



Why Pinning?

Scenario 2: NUMA

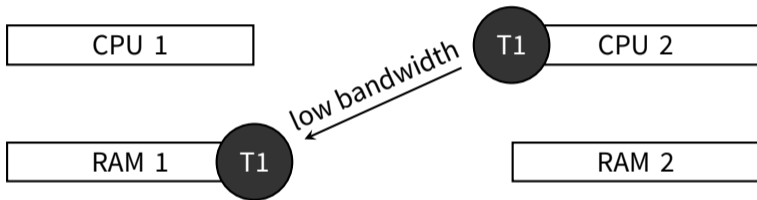
NUMA: Non-Uniform Memory Access, ie memory performance depends on relative location.



Why Pinning?

Scenario 2: NUMA

NUMA: Non-Uniform Memory Access, ie memory performance depends on relative location.

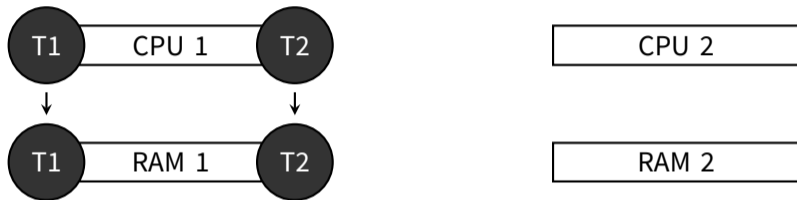


Important

All modern CPUs are NUMA architectures; might even have more than one NUMA domain!
Memory is actually allocated on initialisation, use same parallel configuration as consumer.
There will be no automatic migration.

Why Pinning?

Scenario 3: Sharing Resources



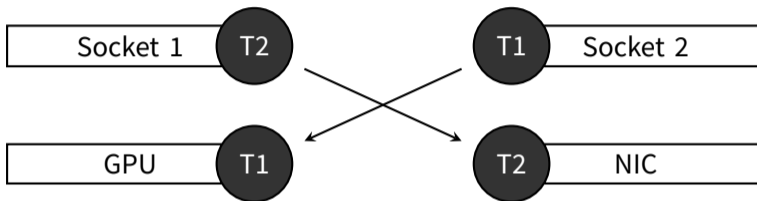
In some instances resources might be shared

- Hardware Threads (HWT) on a core might share computational units.
- Cores on a socket might share memory bandwidth, caches, ...

This can lead to sub-optimal performance by leaving some parts idle and others saturated. The inverse *might also be true*, eg it might be beneficial to share caches for read-only data.

Why Pinning?

Scenario 4: Specialisation



- Accelerators/network interfaces might be attached to a specific socket.
- If tasks/threads have specialised jobs, like MPI communication, ...
- ...scheduling them close to the relevant hardware can improve performance.
- Again: Beware the context switch.

This Talk

- ✓ Motivation: Suboptimal and/or unpredictable performance
- ✓ Definition: What is pinning?
- ✓ Mechanism: Why does it improve performance?
- Learn to know your hardware.
- How to pin your processes.
- How to bind your threads.

Exploring a Node

```
> ml hwloc
> hwloc-ls # IMPORTANT: Run this on the *compute node*, eg via srun!
Machine (754GB total)
  Package L#0
    NUMANode L#0 (P#0 376GB)
    L3 L#0 (28MB)
      L2 L#0 (1024KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0
        PU L#0 (P#0)
        PU L#1 (P#40)
      L2 L#1 (1024KB) + L1d L#1 (32KB) + L1i L#1 (32KB) + Core L#1
        PU L#2 (P#1)
        PU L#3 (P#41)
  [...]
  HostBridge
  PCIBridge
    PCI 3b:00.0 (InfiniBand)
    Net "ib0"
    OpenFabrics "mlx5_0"
  Package L#1
    NUMANode L#1 (P#1 378GB)
    L3 L#1 (28MB)
  [...]
```

[hwloc documentation](#)

Exploring a Node

ASCII Art Edition

```
> hwloc-ls --output-format ascii # IMPORTANT: Run this on the *compute node*, eg via srun!
```

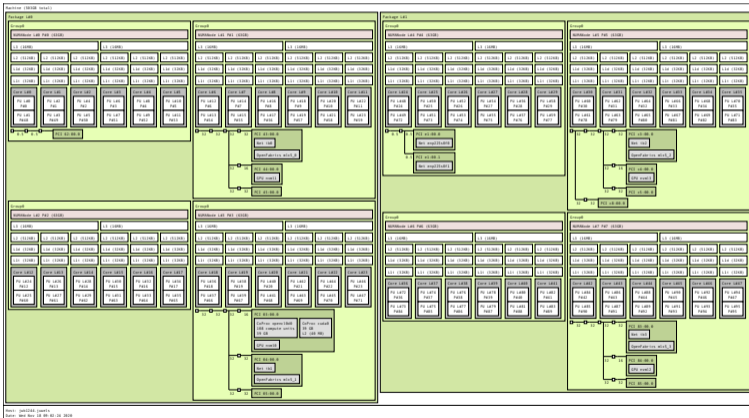
```
+-----+
| Machine (504GB total)
+-----+
| Package L#0
+-----+
| | NUMANode L#0 P#0 (252GB)
+-----+
| | L3 (16MB) | ... | L3 (16MB)
+-----+
| | L2 (512KB) | | L2 (512KB) | | L2 (512KB) | | L2 (512KB) | | L2 (512KB) |
| | L1d (32KB) | | L1d (32KB) | | L1d (32KB) | | L1d (32KB) | | L1d (32KB) |
+-----+
| | Core L#0 | | Core L#1 | | Core L#2 | | Core L#21 | | Core L#22 | | Core L#23 | | | | | | |
| | +-----+ | | +-----+ | | +-----+ | | +-----+ | | +-----+ | | +-----+ |
| | | PU L#0 | | | PU L#2 | | | PU L#4 | | | PU L#42 | | | PU L#44 | | | PU L#46 |
| | | PU L#1 | | | PU L#3 | | | PU L#5 | | | PU L#43 | | | PU L#45 | | | PU L#47 |
| | +-----+ | | +-----+ | | +-----+ | | +-----+ | | +-----+ | | +-----+ |
+-----+
```



Exploring a Node

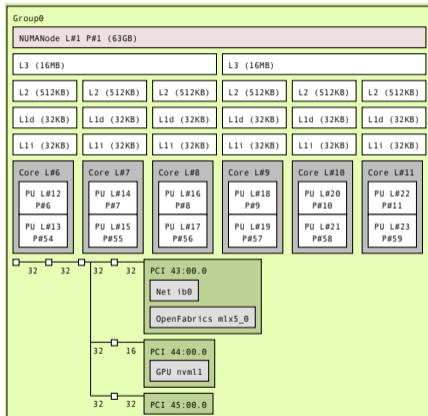
Accelerators and Network Devices

`hwloc-ls --output-format=pdf > node.pdf`



Exploring a Node

Accelerators and Network Devices



Options for Binding

Usually, a hybrid model is used: MPI tasks \times threads (OpenMP/threads/...)

Processes

- Resource Managers: SLURM, ...
- MPI implementations: OpenMPI, PSMPI, ...
- Linux: taskset, numactl, ...
- HWLoc CLI tools

Threads

- OpenMP Environment variables (if used)
- Linux Kernel API
- OpenMP API (if used)
- HWLoc API

Processes: SLURM

Bind

- `--cpu-bind=[options]` Enable binding
 - `verbose` Print binding masks.
 - `cores|threads` Use preset masks.
 - `rank` Bind tasks to CPU IDs matching to task rank.
 - `rank_1dom` Like rank, but distribute across NUMA domains.
 - `mask_cpu=0x..` List of bit masks, can be generated by `hwloc` tools.

Note: binding a process with threads still allows migration between the available HWT.

Defaults

On JUWELS `--cpu-bind=threads` is the default since 22.05; before it was set to ranks.

Warning

SLURM might still generate **bad distributions**, see examples later on.

Processes: SLURM

Distribute

`-N n -n t -c k` Request n nodes for t tasks \times k CPUs per task

`--distribution=L:M:N` Distribute tasks across

`L=block|cyclic` Nodes

`M=block|cyclic|fcyclic` Sockets

`N=block|cyclic|fcyclic` HWT

where

`block` keep tasks as close together as possible

`fcyclic` round-robin distribution, threads round-robin

`cyclic` round-robin distribution, threads close

When a task use more than one CPU, `cyclic` will allocate all as a group within the one socket if possible while `fcyclic` would distribute across sockets.

[slurm documentation](#)

Processes: SLURM

Examples: Single-node

System JUWELS GPU

Node 2 sockets \times 20 cores \times 2 HWT

Request 1 node with 8 tasks \times 3 CPUs

Goal: Optimise for using as much of the hardware as possible, assuming the application does not benefit from co-locating tasks.



`bind=rank`

- ✗ Task 6 is split over two sockets.
- ✗ Uneven load across sockets.

Processes: SLURM

Examples: Single-node

System JUWELS GPU

Node 2 sockets \times 20 cores \times 2 HWT

Request 1 node with 8 tasks \times 3 CPUs

Goal: Optimise for using as much of the hardware as possible, assuming the application does not benefit from co-locating tasks.

0	0	2	4	4	6	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	1	3	5	5	7	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
0	2	2	4	6	6	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	3	3	5	7	7	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

bind=rank_ldom

- ✗ Tasks share cores
- ✗ SMT used before filling HWT

Processes: SLURM

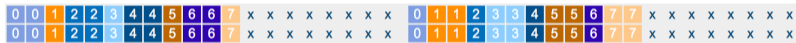
Examples: Single-node

System JUWELS GPU

Node 2 sockets \times 20 cores \times 2 HWT

Request 1 node with 8 tasks \times 3 CPUs

Goal: Optimise for using as much of the hardware as possible, assuming the application does not benefit from co-locating tasks.



bind=cores

X Tasks split over sockets.

Each requested CPU acquires a full physical core.

Processes: SLURM

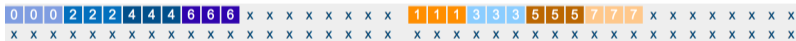
Examples: Single-node

System JUWELS GPU

Node 2 sockets \times 20 cores \times 2 HWT

Request 1 node with 8 tasks \times 3 CPUs

Goal: Optimise for using as much of the hardware as possible, assuming the application does not benefit from co-locating tasks.



bind=threads

✓ Goal achieved

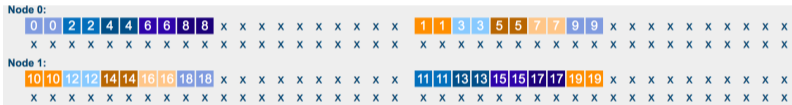
Processes: SLURM

Examples: Multi-node

System JUWELS GPUs

Node 2 sockets \times 20 cores \times 2 HWT

Request 2 nodes with 20 tasks \times 2 CPUs



bind=threads

distribution=block:cyclic:fcyclic

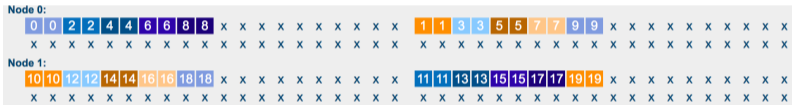
Processes: SLURM

Examples: Multi-node

System JUWELS GPUs

Node 2 sockets \times 20 cores \times 2 HWT

Request 2 nodes with 20 tasks \times 2 CPUs



bind=threads

distribution=block:cyclic:fcyclic

Pinning Visualisation

All images made with [PinningWebtool](#) **Not yet updated** to recent SLURM changes.

Processes: SLURM

Examples: Advanced Usage

System JUWELS Booster: NIC/GPUs attached to NUMA domains 1, 3, 5, 7

Goal 4 dedicated tasks for driving accelerators and communication each.

Processes: SLURM

Examples: Advanced Usage

System JUWELS Booster: NIC/GPUs attached to NUMA domains 1, 3, 5, 7

Goal 4 dedicated tasks for driving accelerators and communication each.

```
> # Compute masks for all HWT in the relevant NUMA domains
> numa=`hwloc-calc numa:1 numa:3 numa:5 numa:7`
> # Generate masks for the distribution of 8 tasks across these
> mask=`hwloc-distrib 8 --single --taskset --restrict $numa | xargs | tr ' ' ','`
> # Run application
> srun --cpu_bind=verbose,cpu_mask=$mask -N 1 -n 8 -c 1 -- app.exe
```

Processes: SLURM

Examples: Advanced Usage

System JUWELS Booster: NIC/GPUs attached to NUMA domains 1, 3, 5, 7

Goal 4 dedicated tasks for driving accelerators and communication each.

```
> # Compute masks for all HWT in the relevant NUMA domains
> numa=`hwloc-calc numa:1 numa:3 numa:5 numa:7`
> # Generate masks for the distribution of 8 tasks across these
> mask=`hwloc-distrib 8 --single --taskset --restrict $numa | xargs | tr ' ' ','`
> # Run application
> srun --cpu_bind=verbose,cpu_mask=$mask -N 1 -n 8 -c 1 -- app.exe
```

Warning

Masks can be computed by hand, but keeping track of the numbering and bitsets is tedious and errorprone. The numbering scheme may change by: vendor, CPU generation, OS, ...

Processes: SLURM

JUWELS Booster Default

Just use the default if your application does not have special requirements.

```
srun -N 1 -n 4 --gpus=4 --cpu-bind=socket -- app.exe
```

This does the right thing and **also** restricts the tasks' visible GPUs to the closest one.

Threads

- When using threads within tasks, these can use affinity as well.
- Without, threads will be mobile within the task-level masks.
- Consequently, we need to add another level of bindings...
- ...and take care not to conflict with task-level masks.

Threads: OpenMP Environment Variables

`OMP_PROC_BIND=[...]` Inhibit migration, bind threads to

- `true` First location it runs on.
- `spread` Spread over allowable set.
- `close` Block threads together.

`OMP_PLACES=[...]` Bind threads to a set of places

- `threads` Individual hardware threads
- `cores` All HWT of a core
- `sockets` All cores of a socket
- `{1, ...}` List of HWT ids

Migration is still allowed within a `place` when binding is not enabled.

Using `threads` | `cores` | `sockets` with task binding is safe.

OpenMP specification

Summary

- Be aware of your application, we cannot provide a general solution.
- Binding for more performance and more predictability.
- Tools like hwloc allow mapping node topologies.
- High-level settings for performance and portability.
Example: SLURM and OpenMP.
- Low-level tools, eg hwloc-API, for ultimate control.

Summary

- Be aware of your application, we cannot provide a general solution.
- Binding for more performance and more predictability.
- Tools like hwloc allow mapping node topologies.
- High-level settings for performance and portability.
Example: SLURM and OpenMP.
- Low-level tools, eg hwloc-API, for ultimate control.

