P^6

PROPER PINNING PREVENTS PRETTY POOR PERFORMANCE

Mai '24 | T. Hater | JSC

JÜLICH
Forschungszentrum
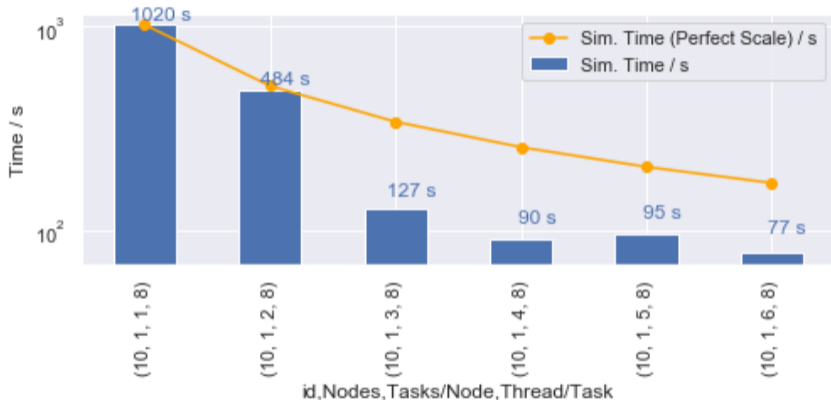
Superlinear Speed-Up?

Superlinear Speed-Up?
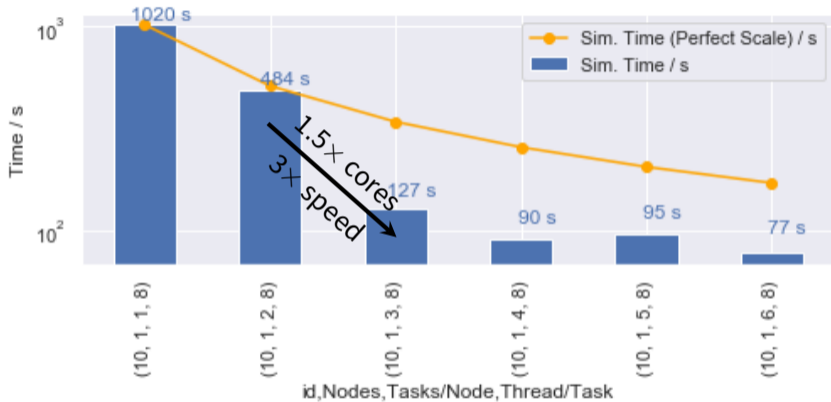
JÜLICH
Forschungszentrum

Superlinear Speed-Up?

No, just a bad baseline…

- Default process placement switched between two cases.

JÜLICH
Forschungszentrum

Superlinear Speed-Up?

No, just a bad baseline…

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

JÜLICH
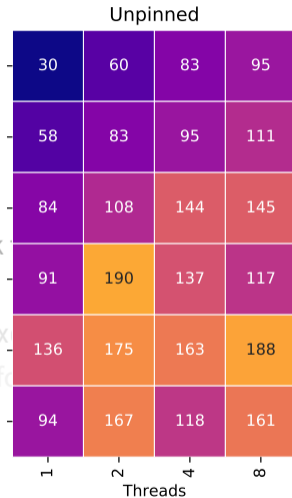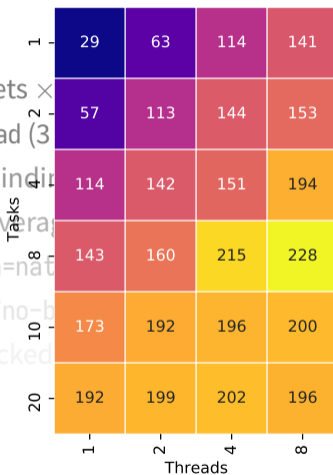Forschungszentrum

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.

JÜLICH
Forschungszentrum

STREAM benchmark

Heavily Optimised for Target Architecture

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

STREAM benchmark

Heavily Optimised for Target Architecture.

Pinned

Unpinned

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

same HW
2× bandwidth

Pinned heatmap (Tasks vs Threads):

Tasks \ Threads	1	2	4	8
1	29	63	114	141
2	57	113	144	153
4	114	142	151	194
8	143	160	215	228
10	173	192	196	200
20	192	199	202	196

Unpinned heatmap (Tasks vs Threads):

Tasks \ Threads	1	2	4	8
1	30	60	83	95
2	58	83	95	111
4	84	108	144	145
8	91	100	137	117
10	136	175	163	188
20	94	167	118	161

JÜLICH
Forschungszentrum

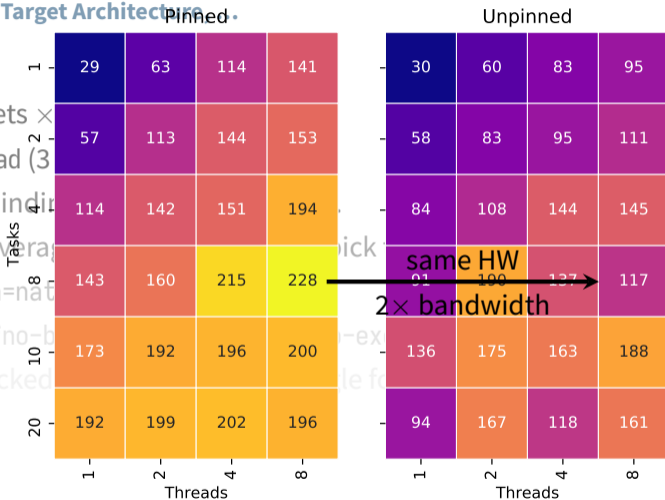# STREAM benchmark
Heavily Optimised for Target Architecture…

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

Pinned

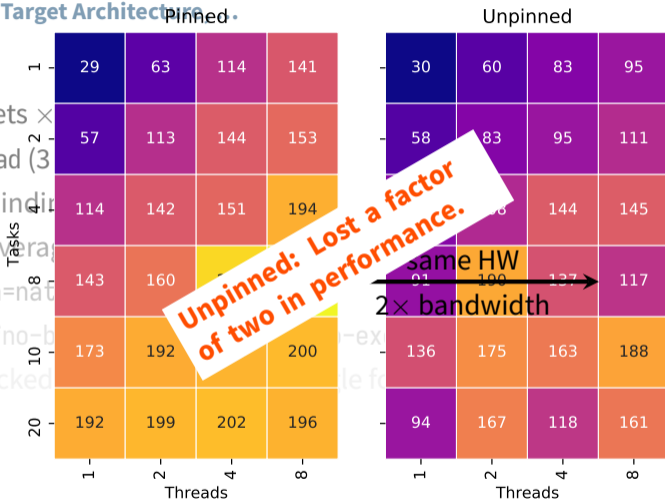| Tasks \ Threads	1	2	4	8
1	29	63	114	141
2	57	113	144	153
4	114	142	151	194
8	143	160		
10	173	192		200
20	192	199	202	196

Unpinned

Tasks \ Threads	1	2	4	8
1	30	60	83	95
2	58	83	95	111
4			144	145
8			137	117
10	136	175	163	188
20	94	167	118	161

Unpinned: Lost a factor of two in performance.

Same HW
2× bandwidth

JÜLICH
Forschungszentrum

What is Pinning?

Also: Binding, Affinity, …

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

JÜLICH
Forschungszentrum

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.

JÜLICH
Forschungszentrum

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.

JÜLICH
Forschungszentrum

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.
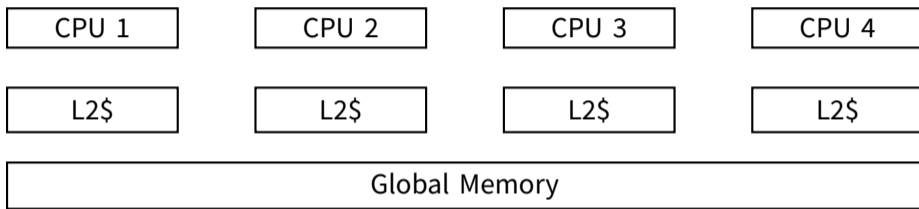- In HPC this is (partially!) handled by the scheduler (SLURM) or MPI.

JÜLICH
Forschungszentrum

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.
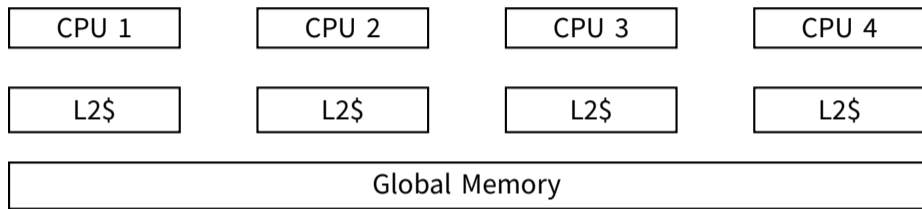- But you can (should?) take control.

JÜLICH
Forschungszentrum

Why Pinning?

A Cartoon CPU

CPU 1	CPU 2	CPU 3	CPU 4
L2$	L2$	L2$	L2$

Global Memory

- Many cores, each with its own
  memory hierarchy.

JÜLICH
Forschungszentrum

Why Pinning?

A Cartoon CPU

CPU 1	CPU 2	CPU 3	CPU 4
L2$	L2$	L2$	L2$

Global Memory

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

JÜLICH
Forschungszentrum

Why Pinning?

A Cartoon CPU

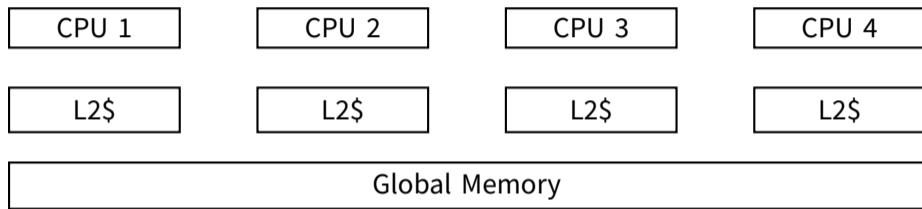| CPU 1	CPU 2	CPU 3	CPU 4
L2$	L2$	L2$	L2$

Global Memory

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

JÜLICH
Forschungszentrum

Why Pinning?

A Cartoon CPU

CPU 1	CPU 2	CPU 3	CPU 4
L2$	L2$	L2$	L2$

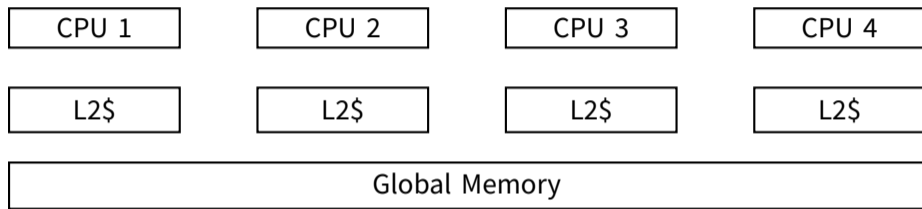| Global Memory |
|---------------|

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

- OS manages allocation,…

JÜLICH Forschungszentrum

Why Pinning?

A Cartoon CPU

CPU 1	CPU 2	CPU 3	CPU 4
L2$	L2$	L2$	L2$

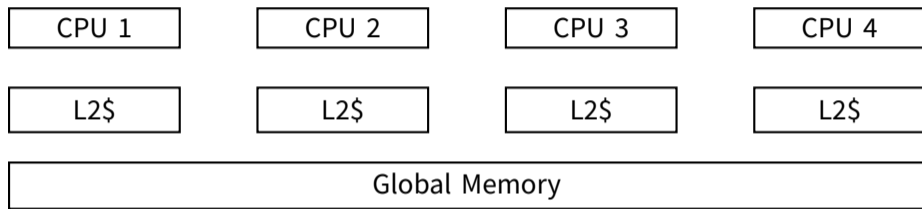| Global Memory |
|---------------|

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

- OS manages allocation,…
- …task placement, and…

JÜLICH
Forschungszentrum

Why Pinning?

A Cartoon CPU

CPU 1	CPU 2	CPU 3	CPU 4
L2\$	L2\$	L2\$	L2\$

Global Memory

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

- OS manages allocation,…
- …task placement, and…
- …swaps tasks in and out.

JÜLICH
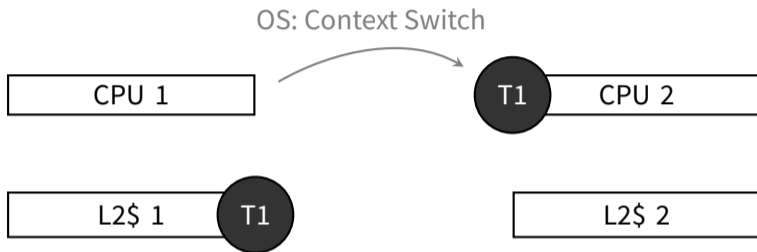Forschungszentrum

Why Pinning?

Scenario 1: Task Migration

Why Pinning?
Scenario 1: Task Migration

Why Pinning?

Scenario 1: Task Migration

JÜLICH
Forschungszentrum
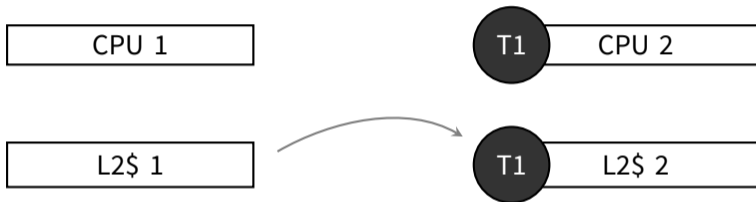
Why Pinning?

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*.

JÜLICH
Forschungszentrum

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.
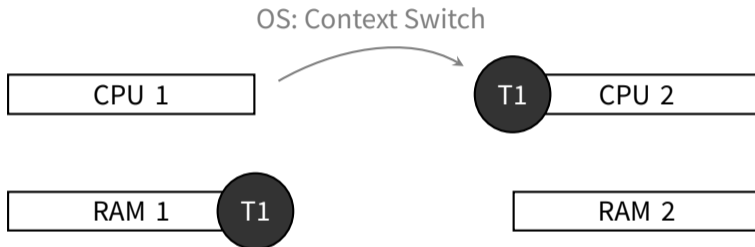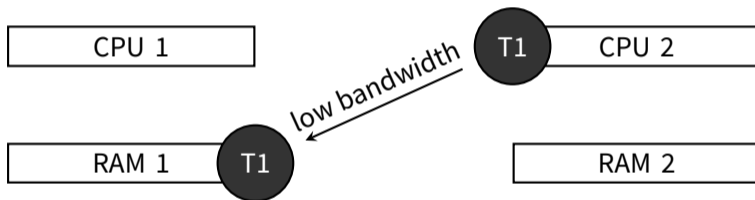
Why Pinning?

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

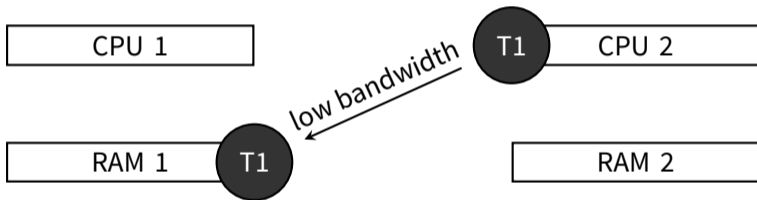All modern CPUs are NUMA architectures; might even have more than one NUMA domain!
Memory is actually allocated on initialisation, use same parallel configuration as consumer.
There will be no automatic migration.

Why Pinning?

Scenario 3: Sharing Resources
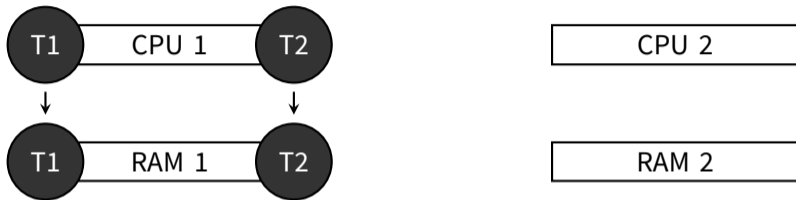
In some instances resources might be shared

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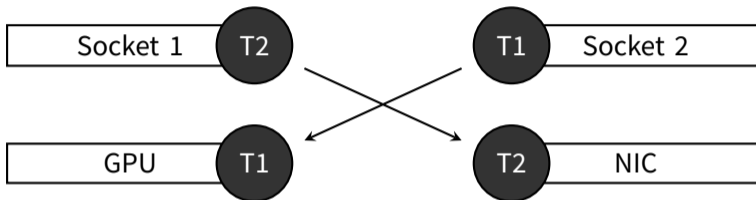
- Accelerators/network interfaces might be attached to a specific socket.
- If tasks/threads have specialised jobs, like MPI communication, …
- …scheduling them close to the relevant hardware can improve performance.
- Again: Beware the context switch.

This Talk

✓ Motivation: Suboptimial 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.

JÜLICH
Forschungszentrum

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

JÜLICH
Forschungszentrum
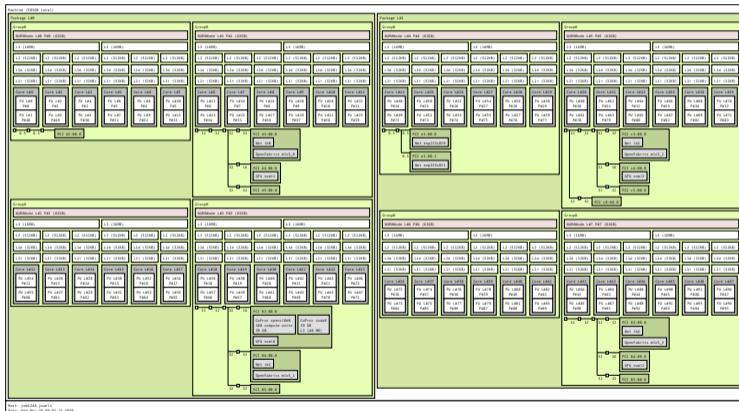
Exploring a Node

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)  | | L2 (512KB)  | | |
| | | L1d (32KB)  | | 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
```

JÜLICH
Forschungszentrum

Exploring a Node

Accelerators and Network Devices

JÜLICH
Forschungszentrum
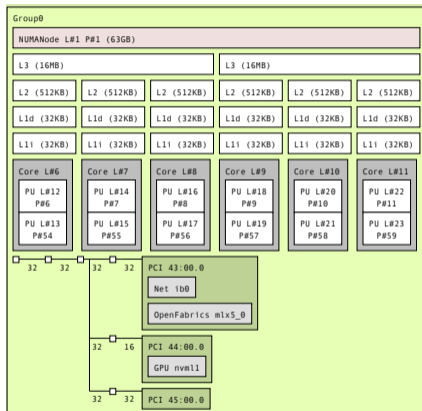
Options for Binding

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

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

JÜLICH
Forschungszentrum

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_ldom`	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.

Warning

SLURM at JSC is currently (v22) in an inconsistent state and will change soon (v23). It is thus highly important to monitor the masks generated for your application and the resulting performance.
Worse, the PinningWebtool **is not yet updated** to recent SLURM changes.

JÜLICH
Forschungszentrum

Processes: SLURM

Distribution

`-N n -n t -c k` Request n nodes for t tasks × 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

The matter of `--exact`

When srun is invoked with `--exact`, SLURM will allocate *as few HWT as possible* to satisfy the requested allocation. Example: `srun -n 6 --exact` will use 6 HWT while `srun -n 6` *may* use 6 *cores*, thus allocating 6 × #*HWT*. NB. That might actually be useful, sometimes.

The crux is in recent versions of SLURM `-c`|`--cpus-per-task` implies `--exact`. You may use `--oversubscribe` to counteract this automatism.

JÜLICH
Forschungszentrum

Processes: SLURM
Distribution II

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

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

<div align="right">

`L=block|cyclic` Nodes
`M=block|cyclic|fcyclic` Sockets
`N=block|cyclic|fcyclic` HWT

</div>

slurm documentation

JÜLICH
Forschungszentrum

Processes: SLURM

Distribution II

`-N n -n t -c k` Request n nodes for t tasks × 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

slurm documentation

Nodes, `default=block`

`block` Close; consecutive task use one node, until full, then the next.

`cyclic` Round-robin; one task per node until all nodes, then start again.

JÜLICH
Forschungszentrum

Processes: SLURM

Distribution II

`-N n -n t -c k` Request n nodes for t tasks × 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

slurm documentation

Sockets, `default=cyclic`

`block` Fill one sockect, then use the next.

`cyclic` Round-robin across sockets.

`fcyclic` Tasks round-robin **and** round-robin cores of each task.

JÜLICH
Forschungszentrum

Processes: SLURM

Distribution II

`-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

slurm documentation

Cores, default=$socket-level

`block`	keep tasks as close together as possible
`cyclic`	Round-robin across CPUs.
`fcyclic`	Tasks round-robin **and** round-robin cores of each task.

JÜLICH
Forschungszentrum

Processes: SLURM

Recommended Usage

```
srun --nodes=<nodes>
     --tasks-per-node=<tasks/node>
     --cpus-per-task=<threads/task>
     --distribution=block:cyclic:cyclic -- ./your_exe <args>
```

JÜLICH
Forschungszentrum

Processes: SLURM

Recommended Usage

```
srun --nodes=<nodes>
     --tasks-per-node=<tasks/node>
     --cpus-per-task=<threads/task>
     --distribution=block:cyclic:cyclic -- ./your_exe <args>
```

Note

--cpus-per-task is required here if you want to set the thread count. Setting SLURM_CPUS_PER_TASK or --cpus-per-task in your sbatch script is no longer supported. Also, remember that --cpus-per-task implies --exact!.

JÜLICH
Forschungszentrum

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.

JÜLICH
Forschungszentrum

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
```

JÜLICH
Forschungszentrum

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, ...

JÜLICH
Forschungszentrum

Processes: SLURM

JUWELS Booster Default

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

```
srun -N 1 -n 4 --gpus=4 --cpus-per-task=12 --cpu-bind=socket -- app.exe
```

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

JÜLICH
Forschungszentrum

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.

JÜLICH
Forschungszentrum

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

JÜLICH
Forschungszentrum

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.

JÜLICH
Forschungszentrum

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.

Happy Pinning
t.hater@fz-juelich.de

JÜLICH
Forschungszentrum