TACC
TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

# OpenMP Affinity Concepts

CSC Summer Institute

**October 9-11, 2023**

Kent Milfeld (TACC)
Emanuele Vitali (CSC)

Slides at tinyurl.com/ adv-omp-2023

# What you will learn

- The basics of Affinity →
  - affinity fundamentals, the affinity mask, and how it is used
- How to acquire affinity mask (and use *amask* utility)
- How to use OpenMP Affinity:

  (proc_bind and places of OpenMP)

# Sections

- Motivation
- Affinity fundamentals, the affinity mask, and how it is used
- OpenMP  Affinity:   proc_bind and places
- Showing Mask with amask utility

# Motivation

- HPC Nodes: many cores, sockets, SMT* (Simultaneous Multi-Threading on a core), NUMA
- HPC User's approach to executing an application is too often:

  Get app to compile (or use site-installed app), prepare input and run:

  `sbatch job` which contains `mpirun app (most likely: 1 core/task)`
- Can HPC performance be improved?

  Need to understand the resource usage (**Yes, it's complicated**):

  Know the Hardware (HW) Architecture

  Evaluate resource usage (htop, remora, etc.)

  Know how to control/map application tasks to resources.
- Specifying where tasks execute through **affinity** (settings) can enhance performance.

So,... it may be necessary to peek inside a node to see what is happening!

*SMT=>Hyper-Threading Technology(intel)/Cluster Multi-Threading(AMD)/SMT(IBM)

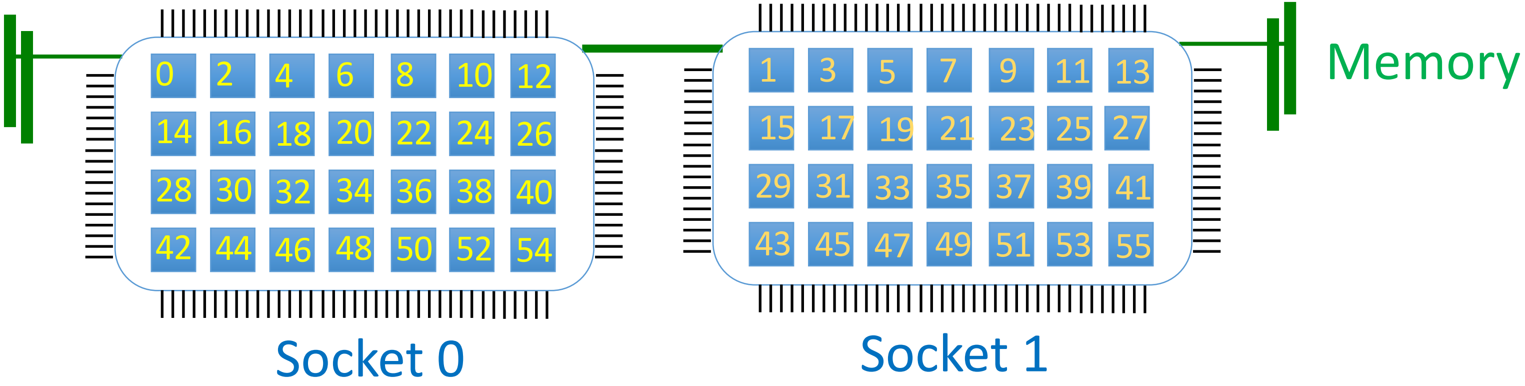
# A peak at Hardware (TACC Frontera Node)

```
$ lscpu

CPU(s):                 56
On-line CPU(s) list:    0-55
Thread(s) per core:     1
Core(s) per socket:     28
Socket(s):              2
NUMA node(s):           2
Model name:             Intel(R) Xeon(R) Platinum 8280 CPU @ 2.70GHz
CPU MHz:                3299.853
CPU max MHz:            4000.0000
CPU min MHz:            1000.0000
L1d cache:              32K
L1i cache:              32K
L2 cache:               1024K
L3 cache:               39424K
NUMA node0 CPU(s):      0,2,4,6,8,10,12,14,16,...,38,40,42,44,46,48,50,52,54
NUMA node1 CPU(s):      1,3,5,7,9,11,13,15,17,...,39,41,43,45,47,49,51,53,55
```

Some information not shown.

# Frontera Node -- cores and memory

2-sockets, 56 Cores, Non-Uniform Memory Access (NUMA), No Hyperthreading



| Socket 0 | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 2 | 4 | 6 | 8 | 10 | 12 |
| 14 | 16 | 18 | 20 | 22 | 24 | 26 |
| 28 | 30 | 32 | 34 | 36 | 38 | 40 |
| 42 | 44 | 46 | 48 | 50 | 52 | 54 |

| Socket 1 | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 7 | 9 | 11 | 13 |
| 15 | 17 | 19 | 21 | 23 | 25 | 27 |
| 29 | 31 | 33 | 35 | 37 | 39 | 41 |
| 43 | 45 | 47 | 49 | 51 | 53 | 55 |

# A peak at Hardware (Lumi/G)

```
$ lscpu
```

```
CPU(s):                          128
On-line CPU(s) list:             0-127
Model name:                      AMD EPYC 7A53 64-Core Processor
Thread(s) per core:              2
Core(s) per socket:              64
Socket(s):                       1
CPU max MHz:                     2000.0000
CPU min MHz:                     1500.0000
L1d cache:                       2 MiB (64 instances)
L1i cache:                       2 MiB (64 instances)
L2 cache:                        32 MiB (64 instances)
L3 cache:                        256 MiB (8 instances)
UMA node(s):                     4
NUMA node0 CPU(s):               0-15,64-79
NUMA node1 CPU(s):               16-31,80-95
NUMA node2 CPU(s):               32-47,96-111
NUMA node3 CPU(s):               48-63,112-127
```

Some information not shown.

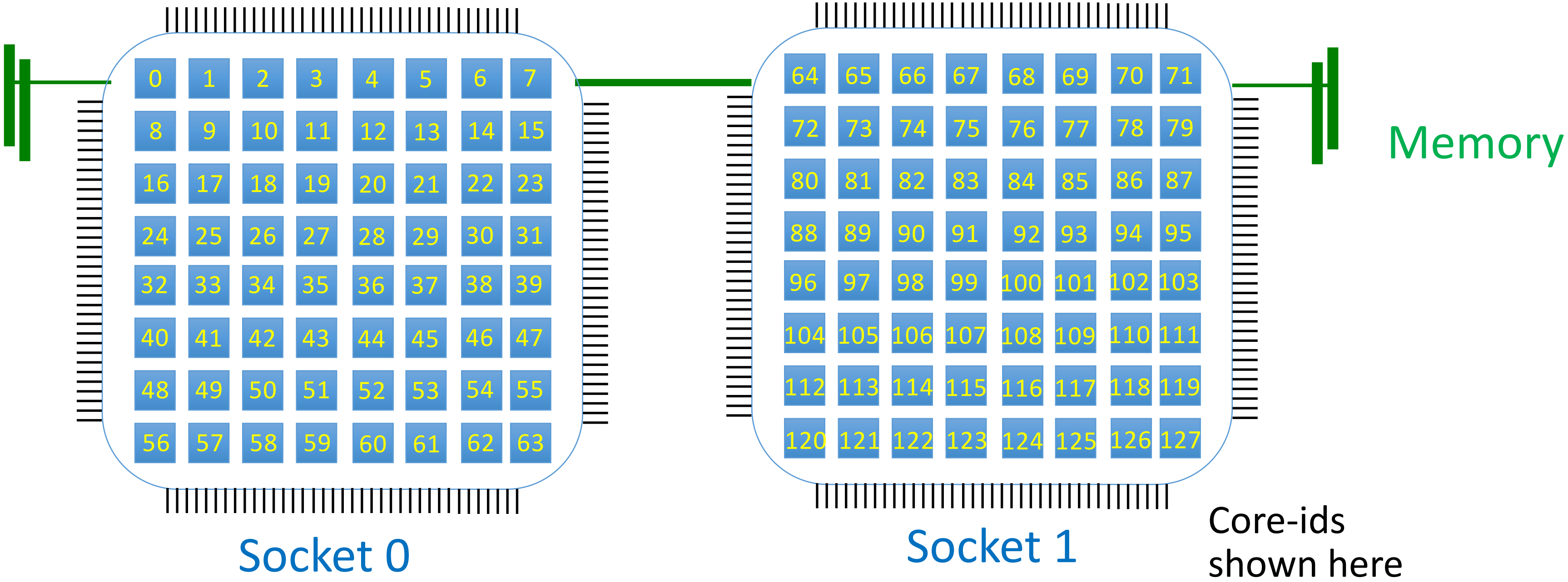
# A peak at Hardware (Lumi/C)

```
$ lscpu

CPU(s):                         256
On-line CPU(s) list:            0-255
Model name:                     AMD EPYC 7763 64-Core Processor
Thread(s) per core:             2
Core(s) per socket:             64
Socket(s):                      2
CPU max MHz:                    2450.0000
CPU min MHz:                    1500.0000
L1d cache:                      4 MiB (128 instances)
L1i cache:                      4 MiB (128 instances)
L2 cache:                       64 MiB (128 instances)
L3 cache:                       512 MiB (16 instances)
NUMA node(s):                   8
NUMA node0 CPU(s):              0-15,128-143
...
NUMA node6 CPU(s):              96-111,224-239
NUMA node7 CPU(s):              112-127,240-255
```

Some information not shown.

# LUMI Node -- cores and memory

2-sockets, 128 Cores, Non-Uniform Memory Access (NUMA), Hyperthreading



|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 |
| 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |

|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 |
| 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 |
| 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 |
| 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
| 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 |
| 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |

Core-ids
shown here

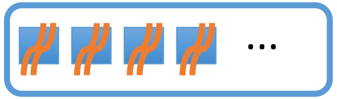
# Example Architecture

## Single Socket (not always simple)

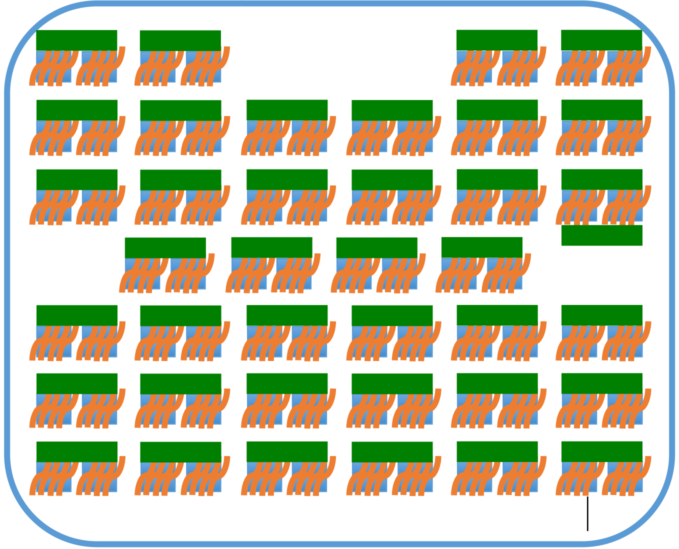
Hyper-Threading (laptop, 2) (KNL, 4 )

2 Cores (a Tile) share a 1MB L2 Cache.

Laptops (simple)

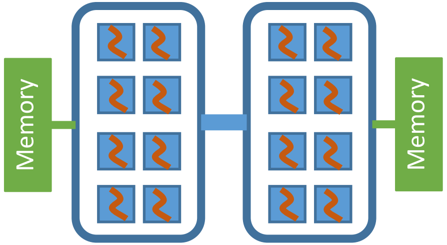


Stampede2 KNL Nodes

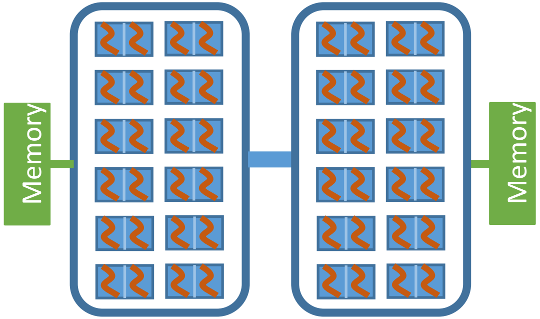


# Unix Architecture from *lscpu*

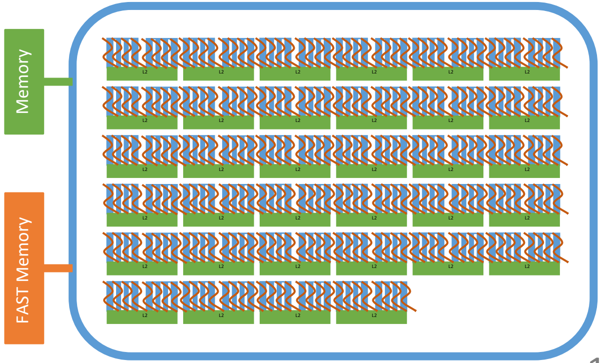
```
sp$ lscpu | grep -i 'core\|thread\|socket'
Thread(s) per core:     1
Core(s) per socket:     8
Socket(s):              2
```



```
ls5%  lscpu | grep -i 'core\|thread\|socket'
Thread(s) per core:     2
Core(s) per socket:     12
Socket(s):              2
```



```
knl$ lscpu | grep -i 'core\|thread\|socket'
Thread(s) per core:     4
Core(s) per socket:     68
Socket(s):              1
```



But more details can be obtained from hwloc.

# Notes from *lscpu* architectural review:

Memory:

On multi-socket nodes it might be best to bind a process to a single socket  to preserve local memory access.

Binding to a single core (when #processes <#cores) may reduce process migration and hence cache refreshes.

# When should affinity be considered?

**Affinity: Determines where application processes can/will execute.**

**Affinity: Needed for these cases:**

- processor count > process count
- processors have local and remote shared memory
- want cache to remain hot through process execution
- multiple IO clients may need separation/positioning

# Motivation

- “Often” applications run fine when using the “normal” number of processes (one process per core)
- 1st Approach to Affinity: Do Nothing, Generally Defaults are “good”
  2nd match process needs to architecture features/resources
  3rd ”pin” processes for limited mobility

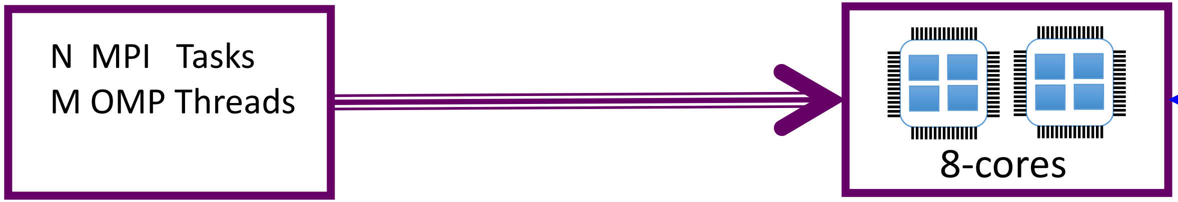
Need Tools to Assess Core Occupation on many-core systems:
Stampede2: KNL -- 68 cores ( 272 “logical” processors), TILES,...
Lonestar6: Milan -- 128 cores ( 128 “logical” processors), 2 SOCKETS,...
Frontera: CLK -- 56 cores ( 56 “logical” processors), 2 SOCKETS,...
LUMI-C: EPYC -- 128 cores ( 256 “logical” processors), 2 SOCKETS,...

# Outline

- Motivation
- Affinity →
  - affinity fundamentals, the affinity mask, and how it is used
- OpenMP Affinity: PROC_BIND and PLACES
- Showing Mask with *amask* utility

# CPU Affinity -- the mask

processes map onto processors



MPI rank-num or thread-num

proc-id (core-id)

A kernel bit mask (array of bits) exists for each process

# of bits = # of processors (proc-ids) set bit→ process can run on proc-id (core-id).

allow rank/thrd-num 0 to run on cores 0-3 →

allow rank/thrd-num 1 to run on cores 4-7 →

| thrd | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | proc-id |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | | | | | |
| 1 | | | | | 1 | 1 | 1 | 1 | |

How are the proc-ids (numbers) assigned to the hardware?

# Hardware – component enumeration

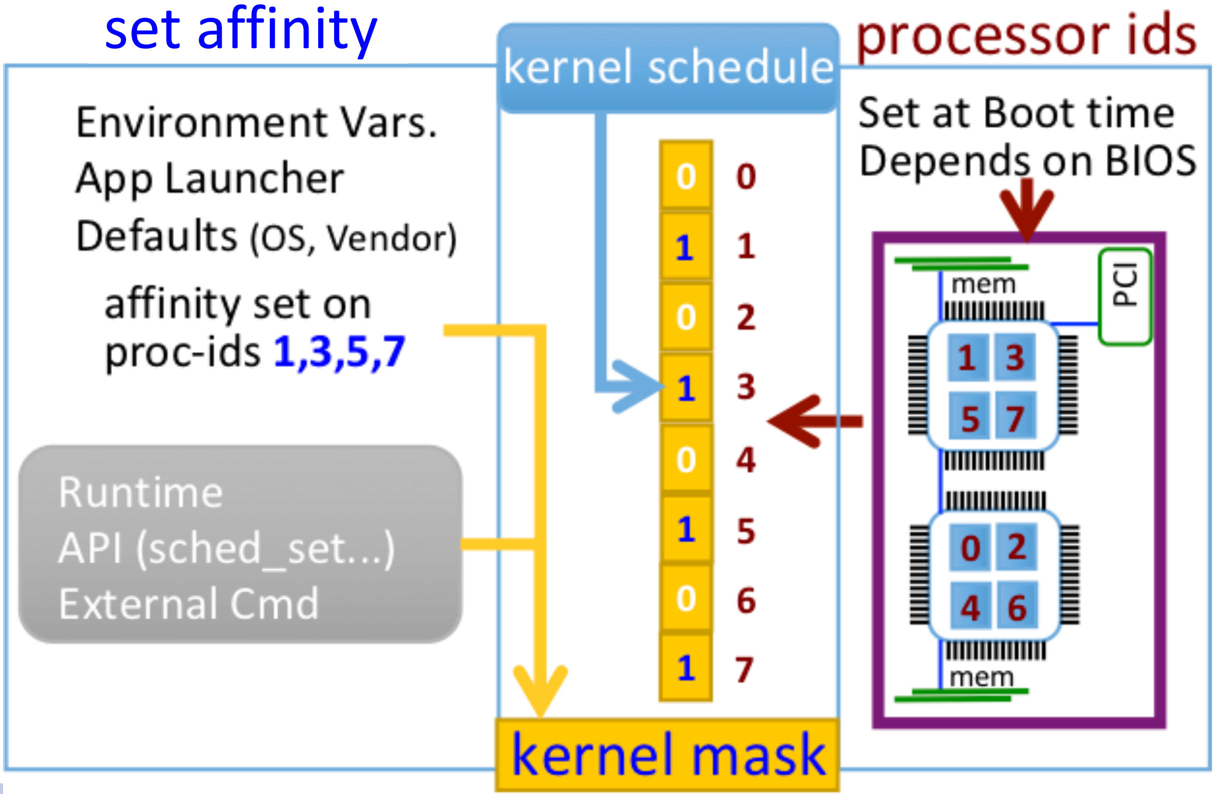
- You may know how many, ... but you cannot assume anything about the numbering, even on an “identical” systems.

- Things may be according to Vendor-OEM BIOS specification, BIOS switch, Site Management, OS level, OpenMP runtime for compiler,...

“Hardware organization is unpredictable”

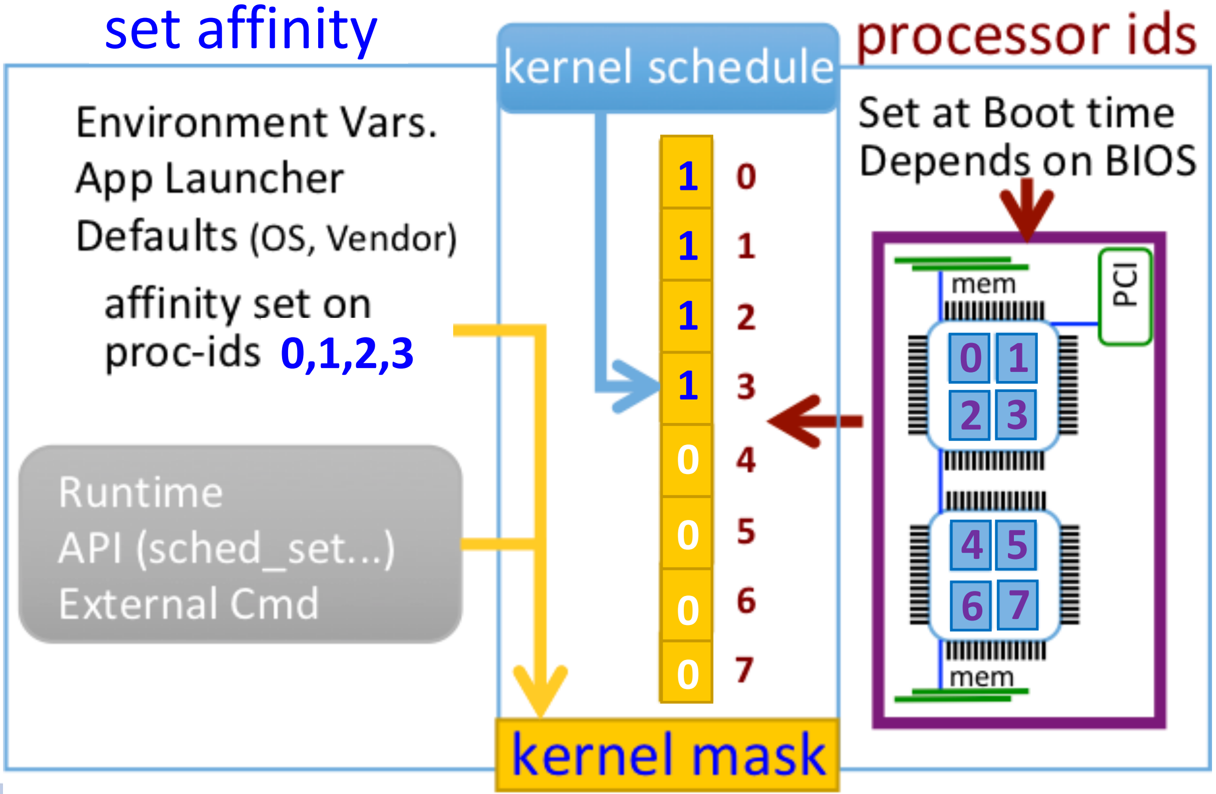
Inria ComPAS' 2013 Brice Goglin

# e.g. setting socket affinity with proc-ids



“Even-Odd Sequence”

# e.g. setting socket affinity with proc-ids



"Even-Odd Sequence"

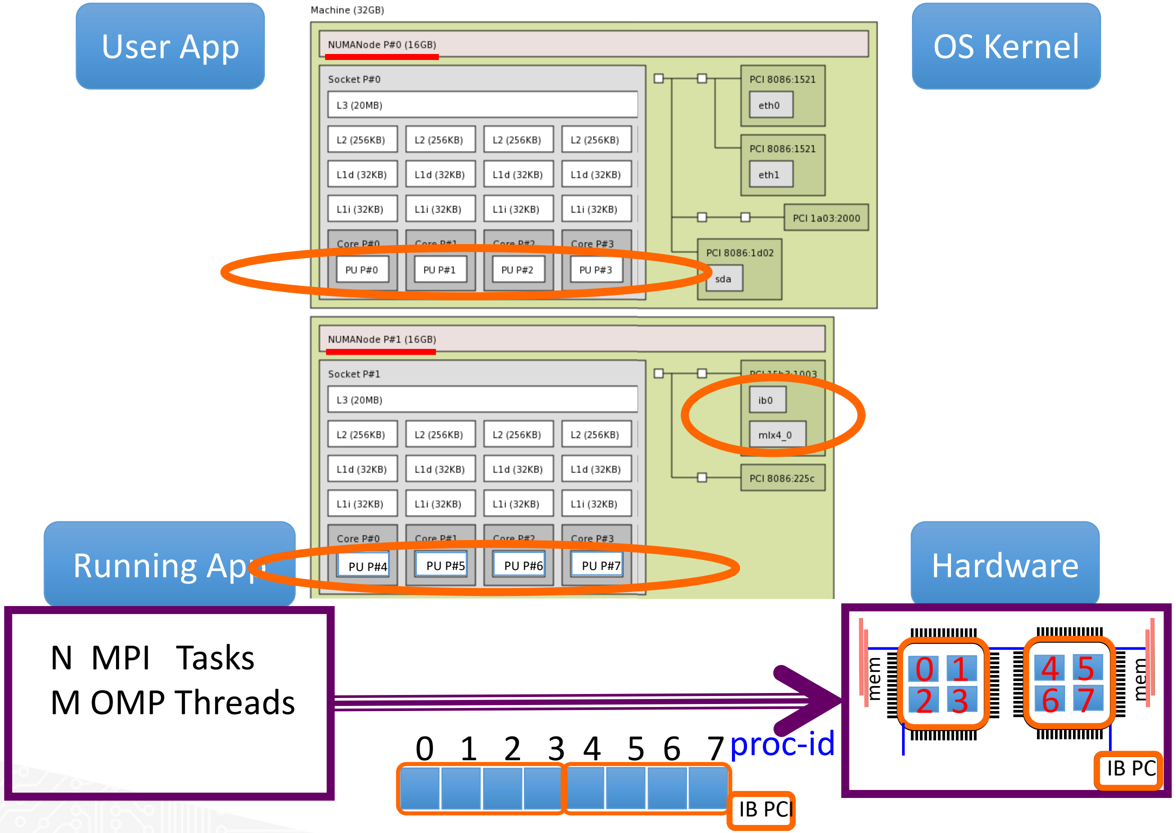
# Setting Affinity

hwloc (lstopo) and /proc/cpuinfo are your friends

For NUMA information, see also numactl -H

# Unix hwloc utility

```
$ lstopo
```



User App

OS Kernel

Running App

Hardware

```
Machine (32GB)
NUMANode P#0 (16GB)
Socket P#0
L3 (20MB)
L2 (256KB)  L2 (256KB)  L2 (256KB)  L2 (256KB)
L1d (32KB)  L1d (32KB)  L1d (32KB)  L1d (32KB)
L1i (32KB)  L1i (32KB)  L1i (32KB)  L1i (32KB)
Core P#0    Core P#1    Core P#2    Core P#3
PU P#0      PU P#1      PU P#2      PU P#3
PCI 8086:1521
eth0
PCI 8086:1521
eth1
PCI 1a03:2000
PCI 8086:1d02
sda
NUMANode P#1 (16GB)
Socket P#1
L3 (20MB)
L2 (256KB)  L2 (256KB)  L2 (256KB)  L2 (256KB)
L1d (32KB)  L1d (32KB)  L1d (32KB)  L1d (32KB)
L1i (32KB)  L1i (32KB)  L1i (32KB)  L1i (32KB)
Core P#0    Core P#1    Core P#2    Core P#3
PU P#4      PU P#5      PU P#6      PU P#7
PCI 15b3:1003
ib0
mlx4_0
PCI 8086:225c
```

N MPI Tasks
M OMP Threads

“Counting Sequence”

# Setting Affinity

## Tasks can be pinned to sets of cores or NUMA nodes

- Each task can have a “mask” of cores it may run on
- A task will can only float to allowed cores (of its mask)
- Many tools for controlling affinity:
  - numactl
  - taskset
  - sched_setaffinity (API, inside code)
  - `OMP_...` `(OpenMP)` & `GOMP_...` (GNU) & `KMP_...` (Intel)
  - MPI implementations have their own affinity tools

# Assigning threads

- How are threads assigned to proc-ids?
  - Build a list of "places" (proc-ids), or use the implementation defaults.
  - Each sequential thread number is assigned to a place.

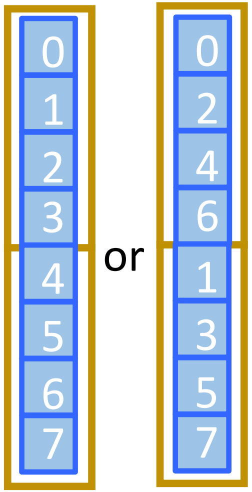
```
#omp parallel num_threads(8)
  printf("%d\n", omp_get_thread_num());
```

A program creates 8 threads sequentially numbered

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

Thread numbers are assigned sequentially to places in place list.

Default place list usually follows sequential numbering from socket to socket. (found in /proc/cpuinfo)



Counting Sequential Even-Odd Sequence

# Unix Architecture from *lscpu*

```
frontera$ lscpu | grep -i 'core\|thread\|Socket'
Thread(s) per core:     1
Core(s) per socket:     28
Socket(s):              2
```

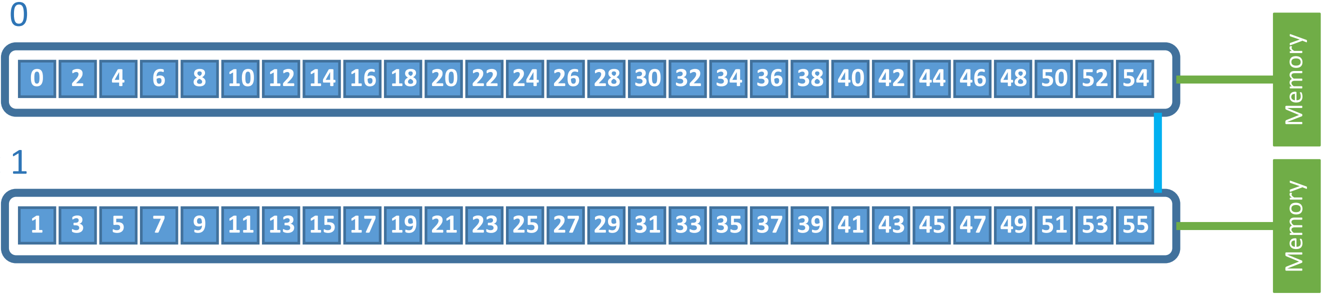
Default Place list: Known to have
Even proc-ids on socket 0.
Odd  proc-ids on socket 1.

## Frontera Compute Node

One would expect the place list to be:

Even on socket 0

Even on socket 1



0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55

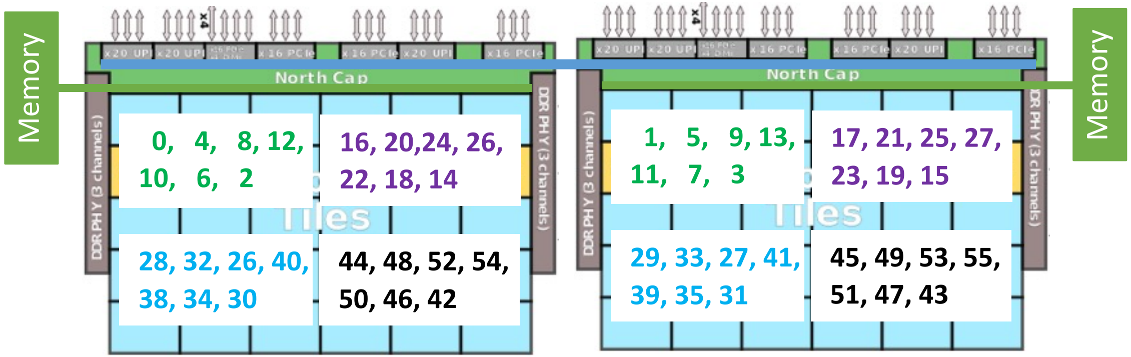
# proc-id #’s on Frontera

socket ▦ core ▢ HW-thread ≀

```
clx% cat /proc/cpuinfo | awk ...
```

```
physical id : 0    core id  :  0    processor :  0
physical id : 0    core id  :  1    processor :  4
physical id : 0    core id  :  2    processor :  8
physical id : 0    core id  :  3    processor : 12
physical id : 0    core id  :  4    processor : 10
physical id : 0    core id  :  5    processor :  6
physical id : 0    core id  :  6    processor :  2
physical id : 0    core id  :  8    processor : 16
physical id : 0    core id  :  9    processor : 20
physical id : 0    core id  : 10    processor : 24
physical id : 0    core id  : 11    processor : 26
physical id : 0    core id  : 12    processor : 22
physical id : 0    core id  : 13    processor : 18
physical id : 0    core id  : 14    processor : 14
physical id : 0    core id  : 16    processor : 28
physical id : 0    core id  : 17    processor : 32
physical id : 0    core id  : 18    processor : 36
physical id : 0    core id  : 19    processor : 40
physical id : 0    core id  : 20    processor : 38
physical id : 0    core id  : 21    processor : 34
physical id : 0    core id  : 22    processor : 30
physical id : 0    core id  : 24    processor : 44
physical id : 0    core id  : 25    processor : 48
physical id : 0    core id  : 26    processor : 52
physical id : 0    core id  : 27    processor : 54
physical id : 0    core id  : 28    processor : 50
physical id : 0    core id  : 29    processor : 46
physical id : 0    core id  : 30    processor : 42
...
```

Note the non-sequential processor (proc-id) numbering.



0, 4, 8, 12, 10, 6, 2

16, 20,24, 26, 22, 18, 14

28, 32, 26, 40, 38, 34, 30

44, 48, 52, 54, 50, 46, 42

1, 5, 9, 13, 11, 7, 3

17, 21, 25, 27, 23, 19, 15

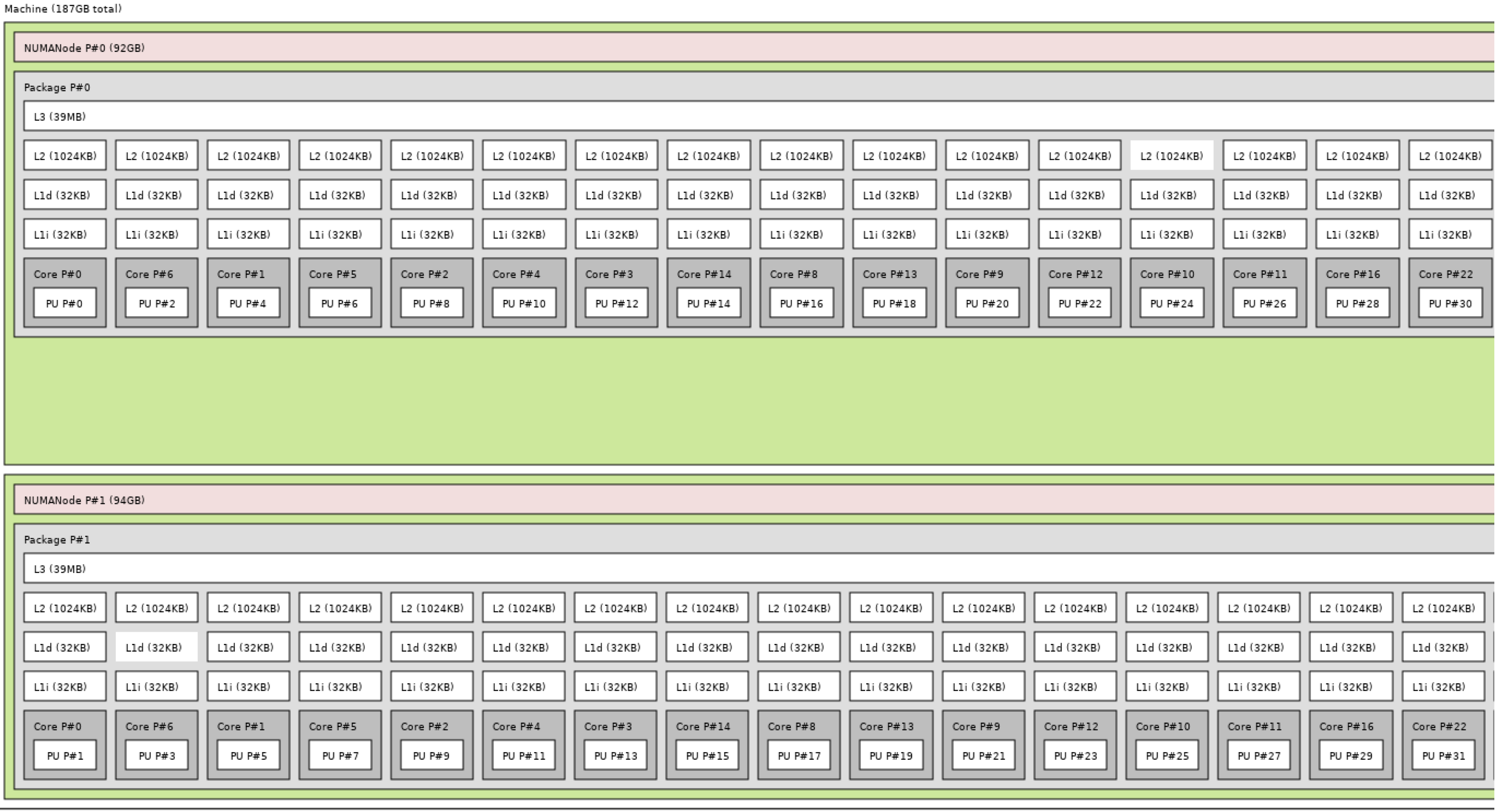
29, 33, 27, 41, 39, 35, 31

45, 49, 53, 55, 51, 47, 43

```
#pragma omp parallel
  printf(“%d\n”, omp_get_thead_num());
```

Thread numbers are assigned sequentially to core-id sequence in /proc/cpuinfo.

# lstopo on Frontera node



Machine (187GB total)

NUMANode P#0 (92GB)

Package P#0

L3 (39MB)

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) |
| L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) |
| L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) |
| Core P#0 | Core P#6 | Core P#1 | Core P#5 | Core P#2 | Core P#4 | Core P#3 | Core P#14 | Core P#8 | Core P#13 | Core P#9 | Core P#12 | Core P#10 | Core P#11 | Core P#16 | Core P#22 |
| PU P#0 | PU P#2 | PU P#4 | PU P#6 | PU P#8 | PU P#10 | PU P#12 | PU P#14 | PU P#16 | PU P#18 | PU P#20 | PU P#22 | PU P#24 | PU P#26 | PU P#28 | PU P#30 |

NUMANode P#1 (94GB)

Package P#1

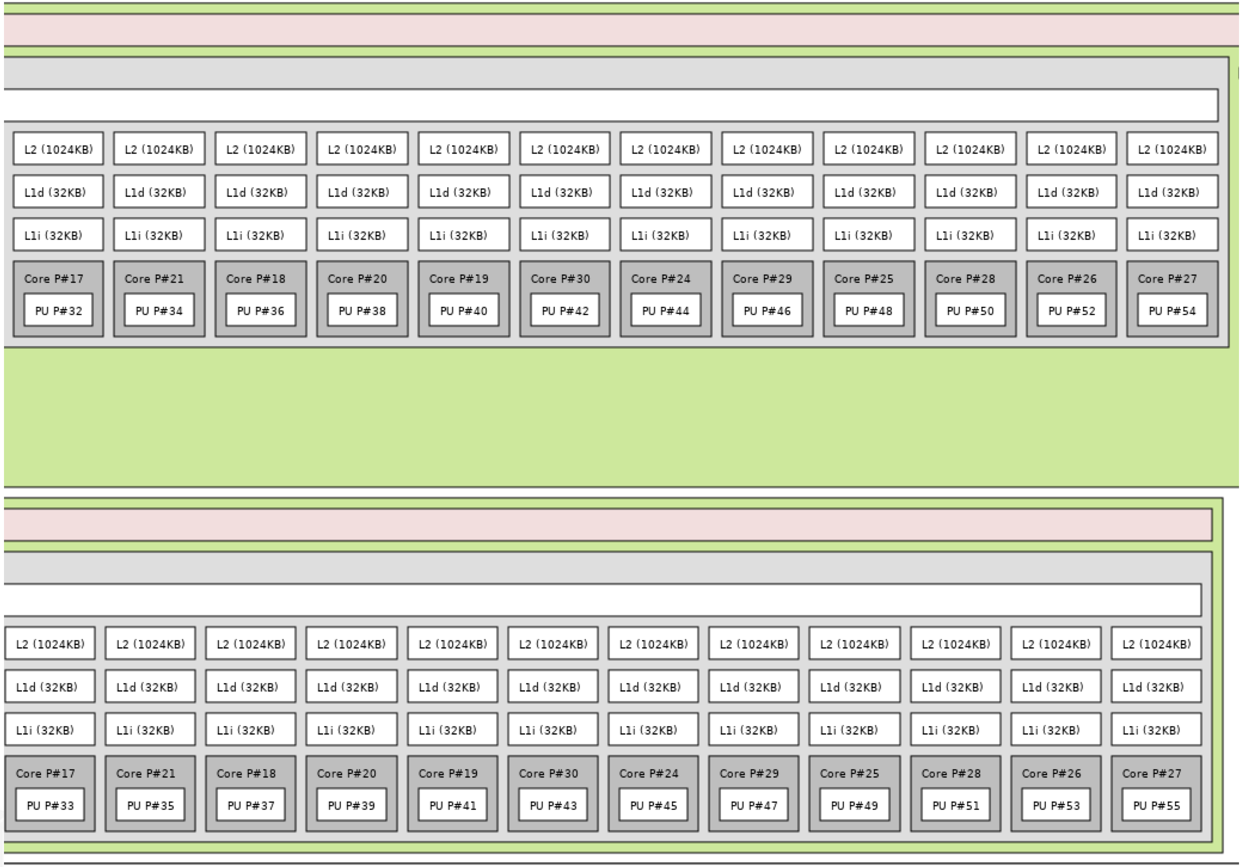
L3 (39MB)

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) |
| L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) |
| L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) |
| Core P#0 | Core P#6 | Core P#1 | Core P#5 | Core P#2 | Core P#4 | Core P#3 | Core P#14 | Core P#8 | Core P#13 | Core P#9 | Core P#12 | Core P#10 | Core P#11 | Core P#16 | Core P#22 |
| PU P#1 | PU P#3 | PU P#5 | PU P#7 | PU P#9 | PU P#11 | PU P#13 | PU P#15 | PU P#17 | PU P#19 | PU P#21 | PU P#23 | PU P#25 | PU P#27 | PU P#29 | PU P#31 |

...

# lstopo on Frontera node (cont. 1)

...



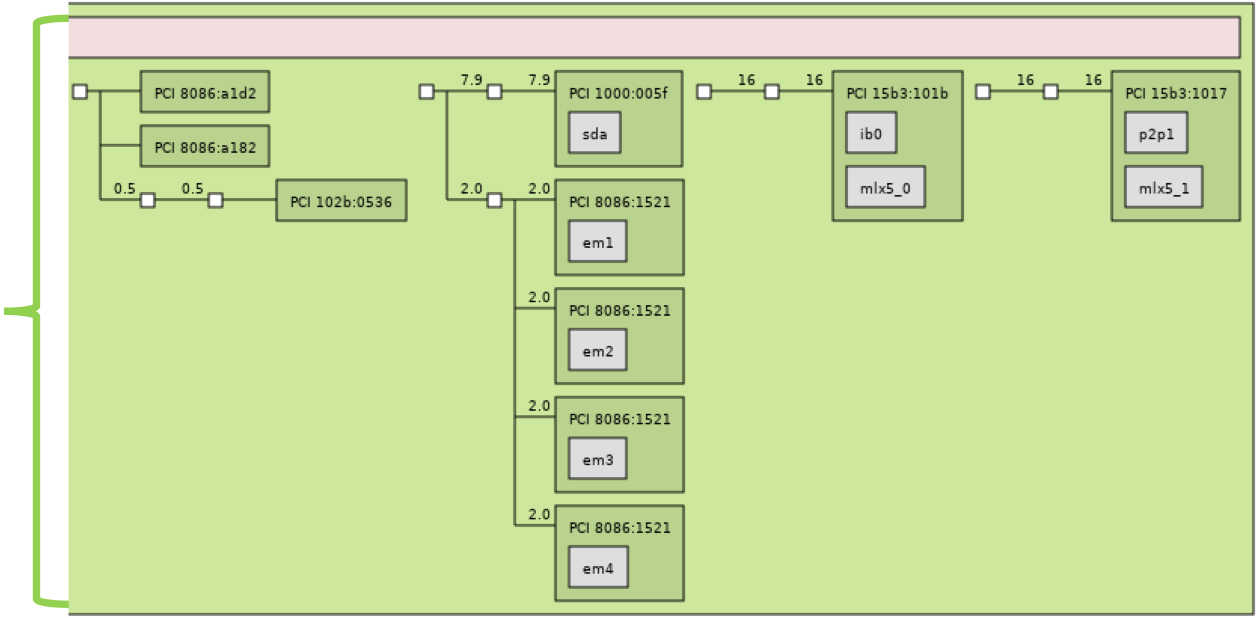
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) |
| L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) |
| L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) |
| Core P#17 | Core P#21 | Core P#18 | Core P#20 | Core P#19 | Core P#30 | Core P#24 | Core P#29 | Core P#25 | Core P#28 | Core P#26 | Core P#27 |
| PU P#32 | PU P#34 | PU P#36 | PU P#38 | PU P#40 | PU P#42 | PU P#44 | PU P#46 | PU P#48 | PU P#50 | PU P#52 | PU P#54 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) | L2 (1024KB) |
| L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB) |
| L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB) |
| Core P#17 | Core P#21 | Core P#18 | Core P#20 | Core P#19 | Core P#30 | Core P#24 | Core P#29 | Core P#25 | Core P#28 | Core P#26 | Core P#27 |
| PU P#33 | PU P#35 | PU P#37 | PU P#39 | PU P#41 | PU P#43 | PU P#45 | PU P#47 | PU P#49 | PU P#51 | PU P#53 | PU P#55 |

...

# lstopo on Frontera node (cont. 2)

…

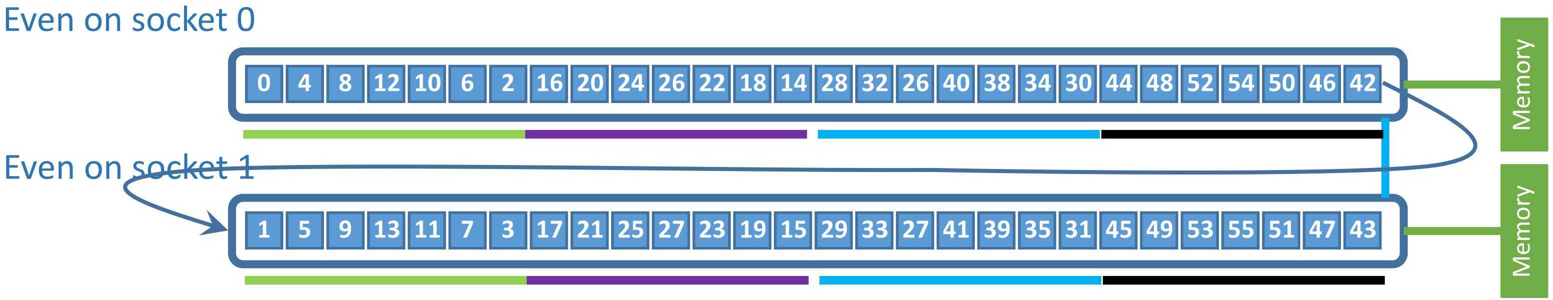


⋮

# Unix Architecture from *lscpu*

## Frontera Compute Node

Think of thread assignment ordering this way.



Even on socket 0

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 4 | 8 | 12 | 10 | 6 | 2 | 16 | 20 | 24 | 26 | 22 | 18 | 14 | 28 | 32 | 26 | 40 | 38 | 34 | 30 | 44 | 48 | 52 | 54 | 50 | 46 | 42 |

Even on socket 1

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | 9 | 13 | 11 | 7 | 3 | 17 | 21 | 25 | 27 | 23 | 19 | 15 | 29 | 33 | 27 | 41 | 39 | 35 | 31 | 45 | 49 | 53 | 55 | 51 | 47 | 43 |

# NUMA nodes on AMD EPYC

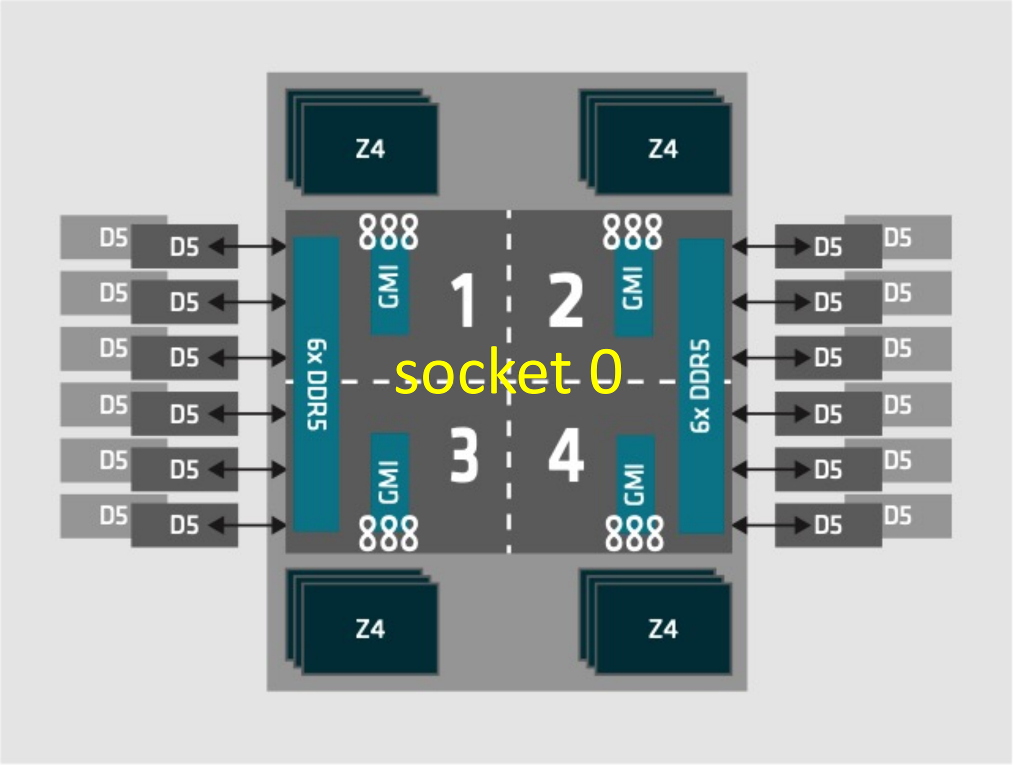


*Figure 7: Dividing the AMD EPYC processor into four NUMA domains can give small performance improvements for some applications*

https://www.amd.com/system/files/documents/4th-gen-epyc-processor-architecture-white-paper.pdf

# LUNI/C Architecture from *lstopo and numactl -H*

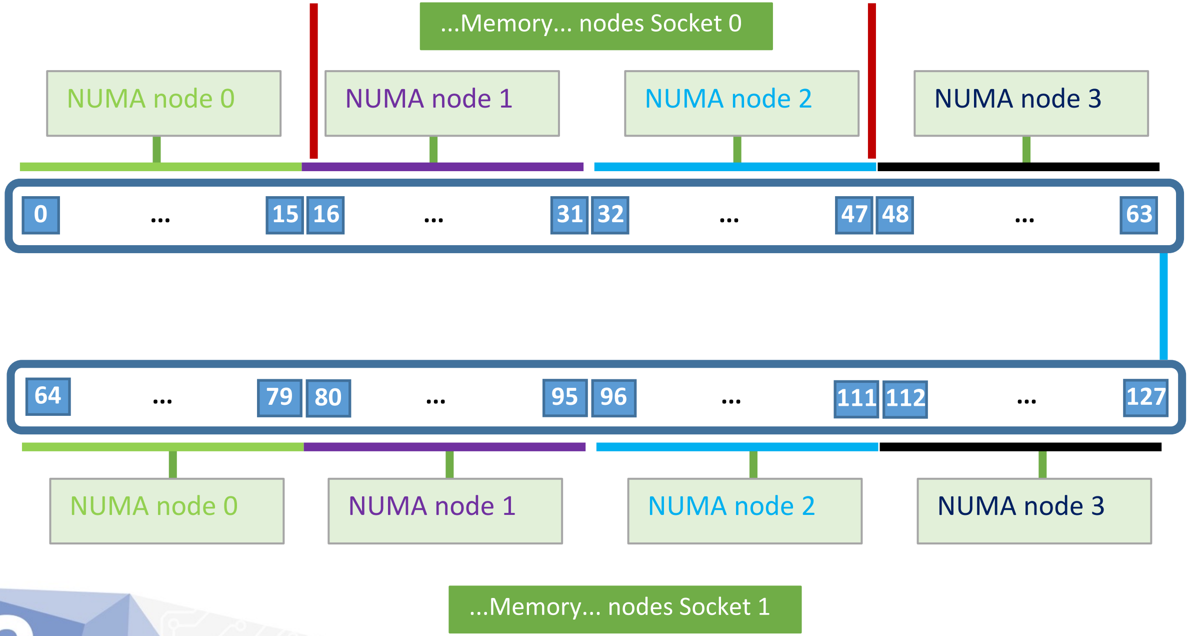
```
$ lstopo
```

```
Machine (251GB total)
  Package L#0
    Group0 L#0
      NUMANode L#0 (P#0 31GB)
      L3 L#0 (32MB)
        L2 L#0 (512KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0
          PU L#0 (P#0)
          PU L#1 (P#128)
        L2 L#1 (512KB) + L1d L#1 (32KB) + L1i L#1 (32KB) + Core L#1
          PU L#2 (P#1)
          PU L#3 (P#129)
        ...
        L2 L#15 (512KB) + L1d L#15 (32KB) + L1i L#15 (32KB) + Core L#15
          PU L#30 (P#15)
          PU L#31 (P#143)
    Group0 L#1
      NUMANode L#1 (P#1 31GB)
      ...
      HostBridge
        PCIBridge
          PCI 41:00.0 (Ethernet)
            Net "nmn0"


    Group0 L#2
      NUMANode L#2 (P#2 31GB)
      L3 L#4 (32MB)
        L2 L#32 (512KB) + L1d L#32 (32KB) + L1i L#32 (32KB) + Core L#32
          PU L#64 (P#32)
          PU L#65 (P#160)
```

# Lumi C Architecture

## LUMI/C Node

Sequential core-id numbers from socket 0 and continuing on socket1



# Outline

- Motivation
- Affinity →
  - affinity fundamentals, the affinity mask, and how it is used
- OpenMP Affinity: **PROC_BIND and PLACES**
- Showing Mask with *amask* utility

# OpenMP Affinity

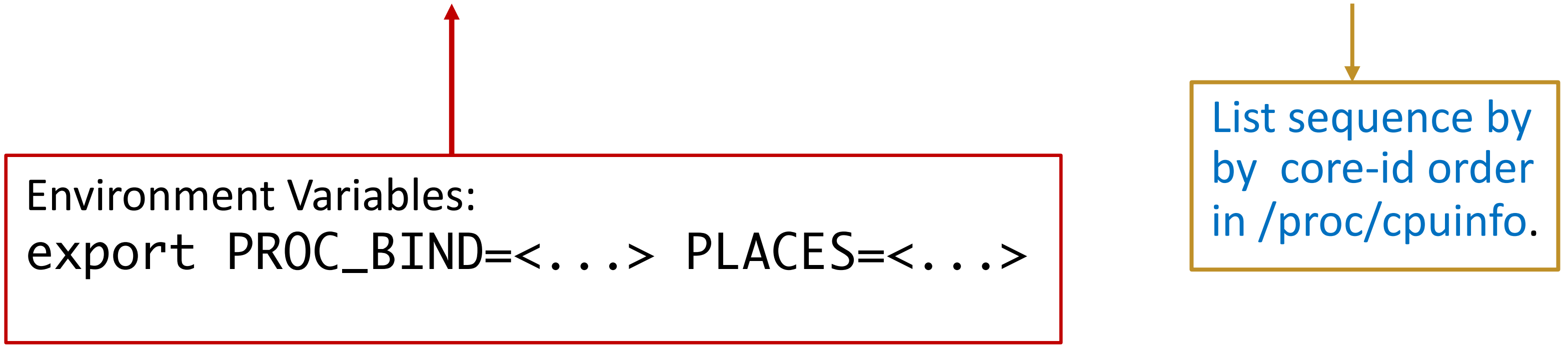
Portable: works with GNU, Intel, IBM, Oracle, …

There are two components to setting affinity:

(after setting the number of threads)

**Distribution Policy**: PROC_BIND policy

**Set Locations:** PLACES User list of proc-ids or abstract set



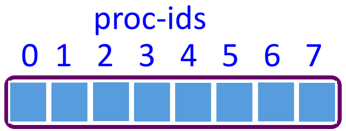
Environment Variables:

```
export PROC_BIND=<...> PLACES=<...>
```

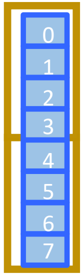
List sequence by by core-id order in /proc/cpuinfo.

# OpenMP Affinity (Policy)

Kernel mask for 8 cores →

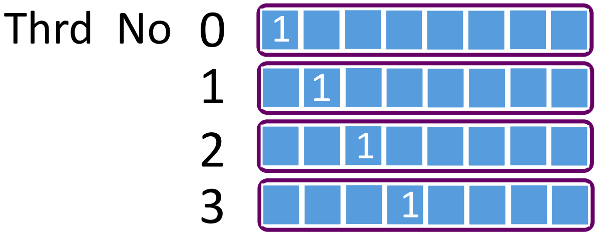


Default Places (/proc/cpuinfo) →

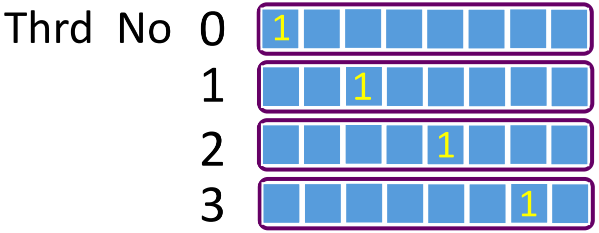


- proc_bind: describes how to distribute threads to places when the number of threads is less than number of proc-ids (cores)

- export OMP_NUM_THREADS=**4**   # (8-core node)
- export OMP_PROC_BIND=close

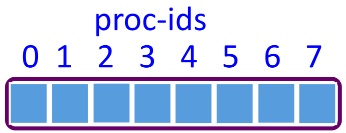


- export OMP_NUM_THREADS=**4**   # (8-core node)
- export OMP_PROC_BIND=spread



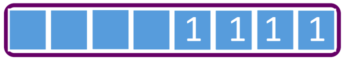
# OpenMP Affinity (User Place List)

Kernel mask for 8 cores →



- place: a set of proc-ids* where a thread can execute

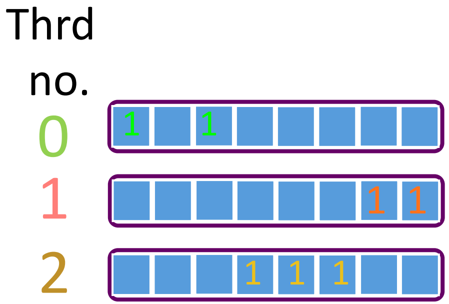
{4, 5, 6, 7}



OMP_NUM_THREADS=3

- place list: list of places assigned to threads in order (for thread no. 0, 1, 2, ...)

{0, 2}, {7, 6} , {3, 4, 5}



*place number = “smallest unit of execution”= proc-id # in Linux systems

# OpenMP Affinity (More on Places)

- Places List

  - export OMP_PLACES=<place_list>

    A place: {0}

    Place List: {0},{1},{2},{3}

    Interval notation: {<place>}:<len>:<stride> e.g. {0}:4:1

    List of proc-ids

  - export OMP_PLACES=<abstract_name>

    Abstract name: sockets, cores*, threads

    Defines a set of places.

    (*cores* default on Frontera)

The precise definitions of the abstract names are implementation defined. An implementation may also add abstract names as appropriate for the target platform.

# OpenMP Affinity (Place assignments)

- In a Parallel region thread numbers 0,...,N-1
  are assigned sequentially to places in the place list.

```
#pragma omp parallel
printf("%d\n",omp_get_thread_num());
```

```
OMP_PLACES=\
"{20},{10},{12},{14,16}"
```

0 ↔ 20
1 ↔ 10
2 ↔ 12
3 ↔ 14,16

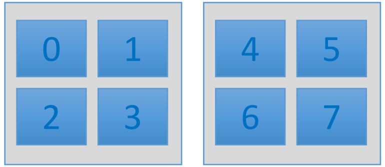
Frontera

OMP_PLACES=cores
OMP_PROC_BIND=close
(uses sequence in
/proc/cpuinfo)

0
4
8
12

[Frontera OMP_PROC_PLACES default is cores]

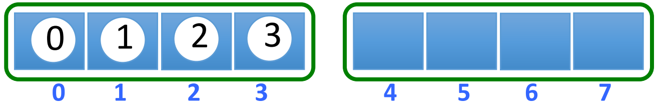
# OpenMP Affinity
# PROC_BIND Distribution



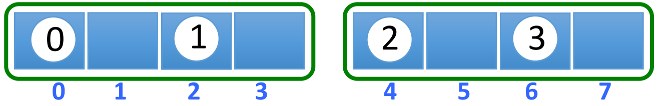
2 sockets x 4 cores

```
export OMP_NUM_THREADS=4
export OMP_PROC_BIND=close

!$omp parallel private(thrd_num);
thrd_num=omp_get_thread_num();
```

```
export OMP_NUM_THREADS=4
export OMP_PROC_BIND=spread

!$omp parallel private(thrd_num);
thrd_num=omp_get_thread_num();
```



COMPACT PACKING



SCATTER PACKING

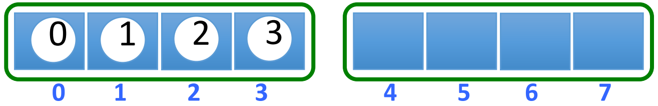
←**thrd_num**

←proc-id

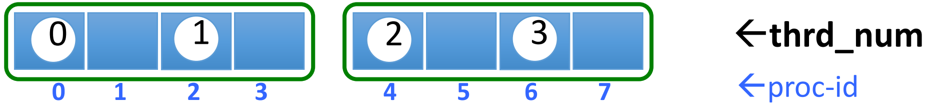
# OpenMP Affinity Places

```
export OMP_NUM_THREADS=4
export OMP_PLACES='{0},{1},{2},{3}'

!$omp parallel private(thrd_num);
thrd_num=omp_get_thread_num();
```

```
export OMP_NUM_THREADS=4
export OMP_PLACES='{0},{2},{4},{6}'

!$omp parallel private(thrd_num);
thrd_num=omp_get_thread_num();
```



COMPACT PACKING

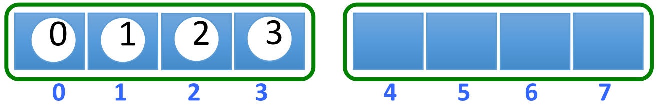


SCATTER PACKING

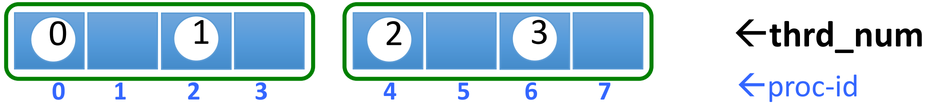
# OpenMP Affinity
# Places → Interval Expression

```
export OMP_NUM_THREADS=4
export OMP_PLACES='{0}:4'

!$omp parallel private(thrd_num);
thrd_num =omp_get_thread_num();
```

```
export OMP_NUM_THREADS=4
export OMP_PLACES='{0}:4:2'

!$omp parallel private(thrd_num);
thrd_num =omp_get_thread_num();
```



COMPACT PACKING

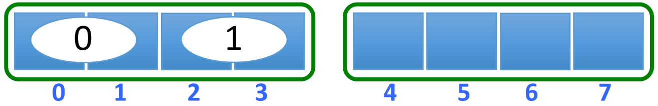


SCATTER PACKING

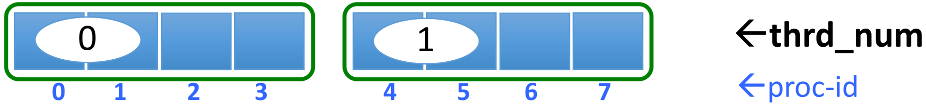
# OpenMP Affinity
# Places → Interval Expression

```
export OMP_NUM_THREADS=2
export OMP_PLACES=‘{0,1}:2:2'

!$omp parallel private(thrd_num);
thrd_num =omp_get_thread_num();
```

```
export OMP_NUM_THREADS=2
export OMP_PLACES=‘{0,1}:2:4'

!$omp parallel private(thrd_num);
thrd_num =omp_get_thread_num();
```



COMPACT PACKING



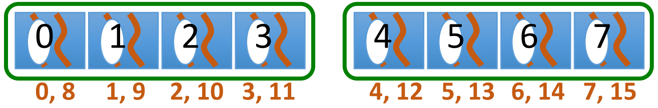
SCATTER PACKING

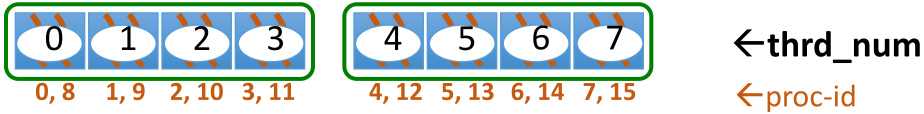
# OpenMP Affinity
# Places → abstract name

## Hyper-Threading Enabled

```
export OMP_NUM_THREADS=8
export OMP_PLACES=threads

!$omp parallel private(thrd_num);
thrd_num =omp_get_thread_num();
```

```
export OMP_NUM_THREADS=8
export OMP_PLACES=cores

!$omp parallel private(thrd_num);
thrd_num =omp_get_thread_num();
```





Binding to Single HW-thread

Binding to Core

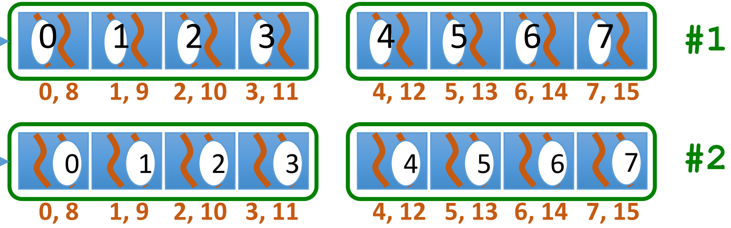
HW-thread

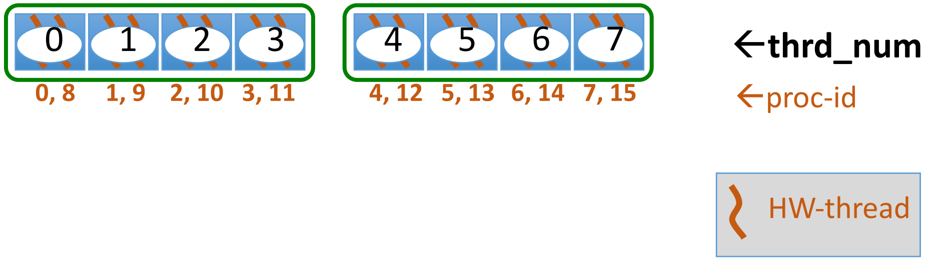
# OpenMP Affinity
# Places expression

## Hyper-Threading Enabled

```
export OMP_NUM_THREADS=8
export OMP_PLACES='{0}:8’   CASE 1
export OMP_PLACES='{8}:8’   CASE 2

!$omp parallel private(thrd_num);
thrd_num =omp_get_thread_num();
```

```
export OMP_NUM_THREADS=8
export OMP_PLACES='{0,8}:8’


!$omp parallel private(thrd_num);
thrd_num =omp_get_thread_num();
```





Binding to Single HW-thread

Binding to Core

# OpenMP Affinity
# Places abstract name

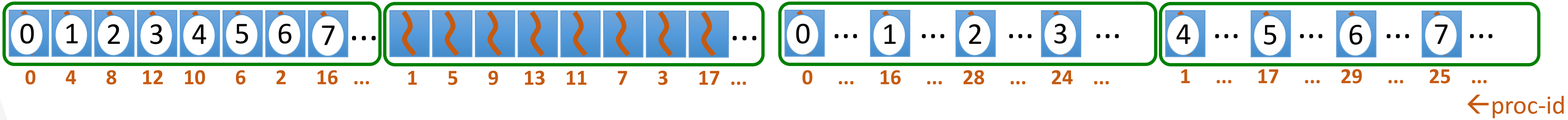
Frontera

even proc-ids: 0,4,8,12,10,6,2,16...

odd proc-ids: 1,5,9,13,11,7,3,17...

```
export OMP_NUM_THREADS=8
export OMP_PLACES=cores
export OMP_PROC_BIND=close

!$omp parallel private(thrd_num);
thrd_num=omp_get_thread_num();
```

```
export OMP_NUM_THREADS=8
export OMP_PLACES=cores
export OMP_PROC_BIND=spread

!$omp parallel private(thrd_num);
thrd_num=omp_get_thread_num();
```



# OpenMP Affinity Places List

## Frontera

even proc-ids: 0,4,8,12,10,6,2,16...

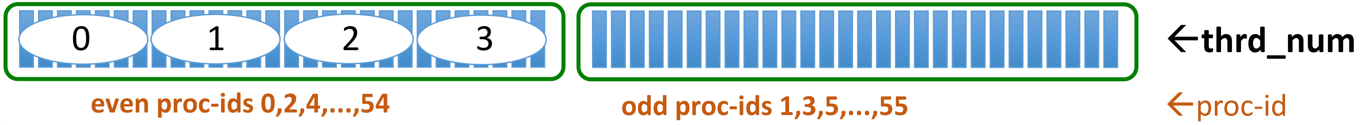
odd proc-ids: 1,5,9,13,11,7,3,17...

evens on socket 1

```
export OMP_NUM_THREADS=4
export OMP_PLACES='{0,2,4,6,8,10,12}:4:14'

!$omp parallel private(thrd_num);
thrd_num=omp_get_thread_num();
```

odds on socket 1

```
export OMP_NUM_THREADS=4
export OMP_PLACES=‘{1,3,5,7,9,11,13}:4:14'

!$omp parallel private(thrd_num);
thrd_num=omp_get_thread_num();
```



# Outline

- Motivation
- Affinity →
    - affinity fundamentals, the affinity mask, and how it is used
- OpenMP  Affinity:   PROC_BIND and PLACES
- Showing Mask with *amask* utility

# amask

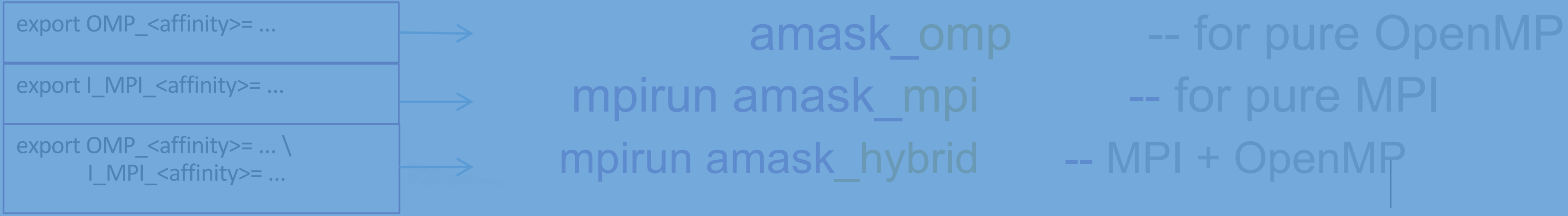
- Report for an affinity env.
  - Executable command: amask_*<type>*   types:{omp, mpi, hybrid}.
  - Can instrument code:  amask_<*type*>() argumentless function calls.

Syntax:

amask_*<type>*   [-h]  [-w#]

help
wait #sec with load

Usage:  module load amask



export OMP_<affinity>= ... → amask_omp -- for pure OpenMP

export I_MPI_<affinity>= ... → mpirun amask_mpi -- for pure MPI

export OMP_<affinity>= ... \
I_MPI_<affinity>= ... → mpirun amask_hybrid -- MPI + OpenMP

# Viewing Affinity mask with amask

Old Stampede: 16-core

export OMP_NUM_THREADS=8 OMP_PROC_BIND=spread

amask_omp

```
         0   proc-id    15
thrd     v              v
 | 0     1000000000000000
 v 1     0010000000000000
   2     0000100000000000
   3     0000001000000000
   4     0000000010000000
   5     0000000000100000
   6     0000000000001000
   7     0000000000000010
```

Bit Mask

```
                proc-id
thrd    |  0       |     10    |
   0    1---------------
   1    --1-------------
   2    ----1-----------
   3    ------1---------
   4    --------1-------
   5    ----------1-----
   6    ------------1---
   7    --------------1-
rank    |  0       |     10    |
```

More Readable

```
                proc-id
thrd    |  0       |     10    |
   0    0---------------
   1    --2-------------
   2    ----4-----------
   3    ------6---------
   4    --------8-------
   5    ----------0-----
   6    ------------2---
   7    --------------4-
rank    |  0       |     10    |
```

Even More Readable

# Frontera

$ OMP_NUM_THREADS=2  amask_omp

NO OpenMP Affinity

```
thrd |    0    |   10    |   20    |   30    |   40    |   50    |
0000 01234567890123456789012345678901234567890123456789012345
0001 01234567890123456789012345678901234567890123456789012345
```

$ OMP_NUM_THREADS=2  OMP_PROC_BIND=spread amask_omp

1 thread per socket

```
thrd |    0    |   10    |   20    |   30    |   40    |   50    |
0000 0-------------------------------------------------------
0001 -1------------------------------------------------------
```

$ OMP_NUM_THREADS=2  OMP_PROC_BIND=close amask_omp

Both threads on 1  socket

```
thrd |    0    |   10    |   20    |   30    |   40    |   50    |
0000 0-------------------------------------------------------
0001 ----4---------------------------------------------------
```

# Frontera

$ OMP_NUM_THREADS=28 \
  OMP_PROC_BIND=close \
  amask_omp

```
thrd
|    0    |   10    |   20    |   30    |   40    |   50    |
0000 0-------------------------------------------------------
0001 ----4---------------------------------------------------
0002 --------8-----------------------------------------------
0003 ------------2-------------------------------------------
0004 ----------0---------------------------------------------
0005 ------6-------------------------------------------------
0006 --2-----------------------------------------------------
0007 ----------------6---------------------------------------
0008 --------------------0-----------------------------------
0009 ------------------------4-------------------------------
0010 --------------------------6-----------------------------
0011 ----------------------2---------------------------------
0012 ------------------8-------------------------------------
0013 --------------4-----------------------------------------
0014 ----------------------------8---------------------------
0015 --------------------------------2-----------------------
0016 ------------------------------------6-------------------
0017 ----------------------------------------0---------------
0018 --------------------------------------8-----------------
0019 ----------------------------------4---------------------
0020 ------------------------------0-------------------------
0021 --------------------------------------------4-----------
0022 ------------------------------------------------8-------
0023 ----------------------------------------------------2---
0024 ------------------------------------------------------4-
0025 --------------------------------------------------0-----
0026 ----------------------------------------------6---------
0027 ------------------------------------------2-------------
```

## /proc/cpuinfo

```
processor :  0
processor :  4
processor :  8
processor : 12
processor : 10
processor :  6
processor :  2
processor : 16
processor : 20
processor : 24
processor : 26
processor : 22
processor : 18
processor : 14
processor : 28
processor : 32
processor : 36
processor : 40
processor : 38
processor : 34
processor : 30
processor : 44
processor : 48
processor : 52
processor : 54
processor : 50
processor : 46
processor : 42
```

# Frontera

```
$ export OMP_NUM_THREADS=4
$ export OMP_PLACES='{0,2,4,6,8,10,12}:4:14'
$ amask_omp
```

All on the same socket  (even proc-ids)

```
thrd |    0    |   10    |   20    |   30    |   40    |   50    |
0000 0-2-4-6-8-0-2-------------------------------------------
0001 --------------4-6-8-0-2-4-6-----------------------------
0002 ----------------------------8-0-2-4-6-8-0---------------
0003 ------------------------------------------2-4-6-8-0-2-4-
```

```
$ export OMP_PLACES='{0,2,4,6,8,10,12}:4:14, {1,3,5,7,9,11,13}:4:14'
$ export OMP_NUM_THREADS=8
$ amask_omp
```

Across sockets

```
thrd |    0    |   10    |   20    |   30    |   40    |   50    |
0000 0-2-4-6-8-0-2-------------------------------------------
0001 --------------4-6-8-0-2-4-6-----------------------------
0002 ----------------------------8-0-2-4-6-8-0---------------
0003 ------------------------------------------2-4-6-8-0-2-4-
0004 -1-3-5-7-9-1-3------------------------------------------
0005 ---------------5-7-9-1-3-5-7----------------------------
0006 -----------------------------9-1-3-5-7-9-1--------------
0007 -------------------------------------------3-5-7-9-1-3-5
```

# What about … hyper-threading?

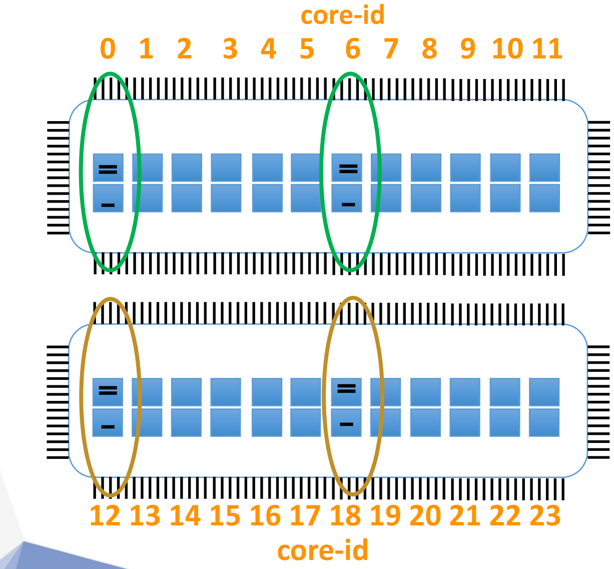
- With Hyper-Threading it is more more reasonable to report the SMTs (hardware threads) together.

# What about hyper-threading…

Will skip in short course.

Hyper-Threaded systems    2 x 12 cores → 48 hardware threads

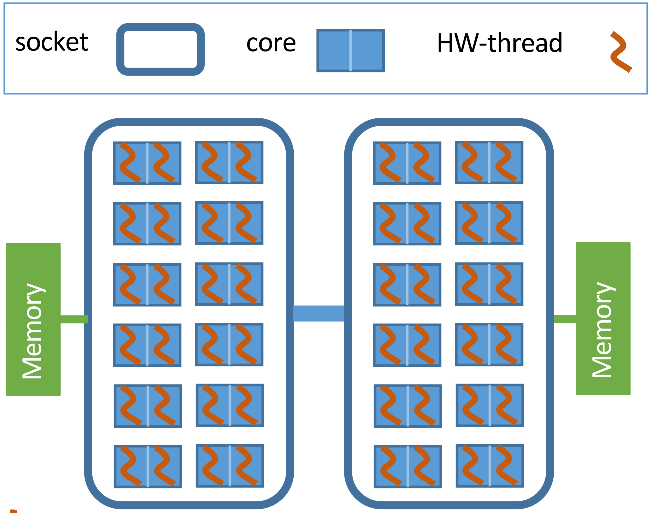
```
$ export OMP_NUM_THREADS=4 OMP_PLACES=cores
$ amask_omp
```



defaults to "core-id" mask when hyper-threading is on

```
thrd |    0    |   10    |   20   ← core-id
0000 0=======================   HW-thread 0
     0-----------------------   HW-thread 1
0001 ======6=================
     ------6-----------------
0002 ============2===========
     ------------2-----------
0003 ==================8=====
     ------------------8-----
```

# proc-id #'s in /cpu/info



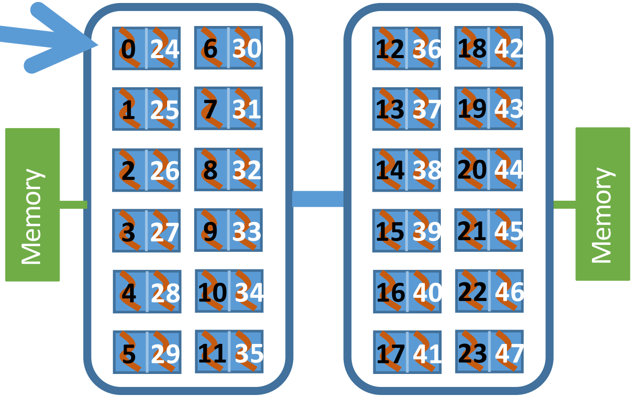
```
ls5%  lscpu | grep -i 'core\|thread\|socket'
Thread(s) per core:     2
Core(s) per socket:     12
Socket(s):              2
```

Hierarchy summary

```
ls5% cat /proc/cpuinfo ...awk* ...

   physical id : 0   core id  : 0   processor : 0
   physical id : 0   core id  : 0   processor : 24
   physical id : 0   core id  : 1   processor : 1
   physical id : 0   core id  : 1   processor : 25
   ...
   physical id : 1   core id  : 0   processor : 12
   physical id : 1   core id  : 0   processor : 36
   physical id : 1   core id  : 1   processor : 13
   ...
```

proc-id

#s



```
ls5% * awk '/processor|core id|physical id/ {arr[j++]=$0};
       END{for(i=0;i<j;i+=3) {printf "%-20s %-20s %-20s\n",
       arr[i+1],arr[i+2],arr[i]} }' /proc/cpuinfo | \
       sed -e 's/[ \t]/ /g' | sort -n -k4,4 -k8,8 -k11,11
```

TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU

TEXAS

The University of Texas at Austin

# Questions!