SEHLOC
2013/12/05
LIDIC - UNSL

Understanding and managing hardware affinities on hierarchical platforms With Hardware Locality (hwloc)

Brice Goglin

Agenda

- Quick example as an Introduction
- Bind your processes
- What's the actual problem?
- Introducing hwloc (Hardware Locality)
- Command-line tools
- Use cases
- Conclusion

1 Quick example as an Introduction

Machines are increasingly complex

Machines are increasingly complex

- Multiple processor sockets
- Multicore processors
- Simultaneous multithreading
- Shared caches
- NUMA
- GPUs, NICs, …
  - Close to some sockets (NUIOA)

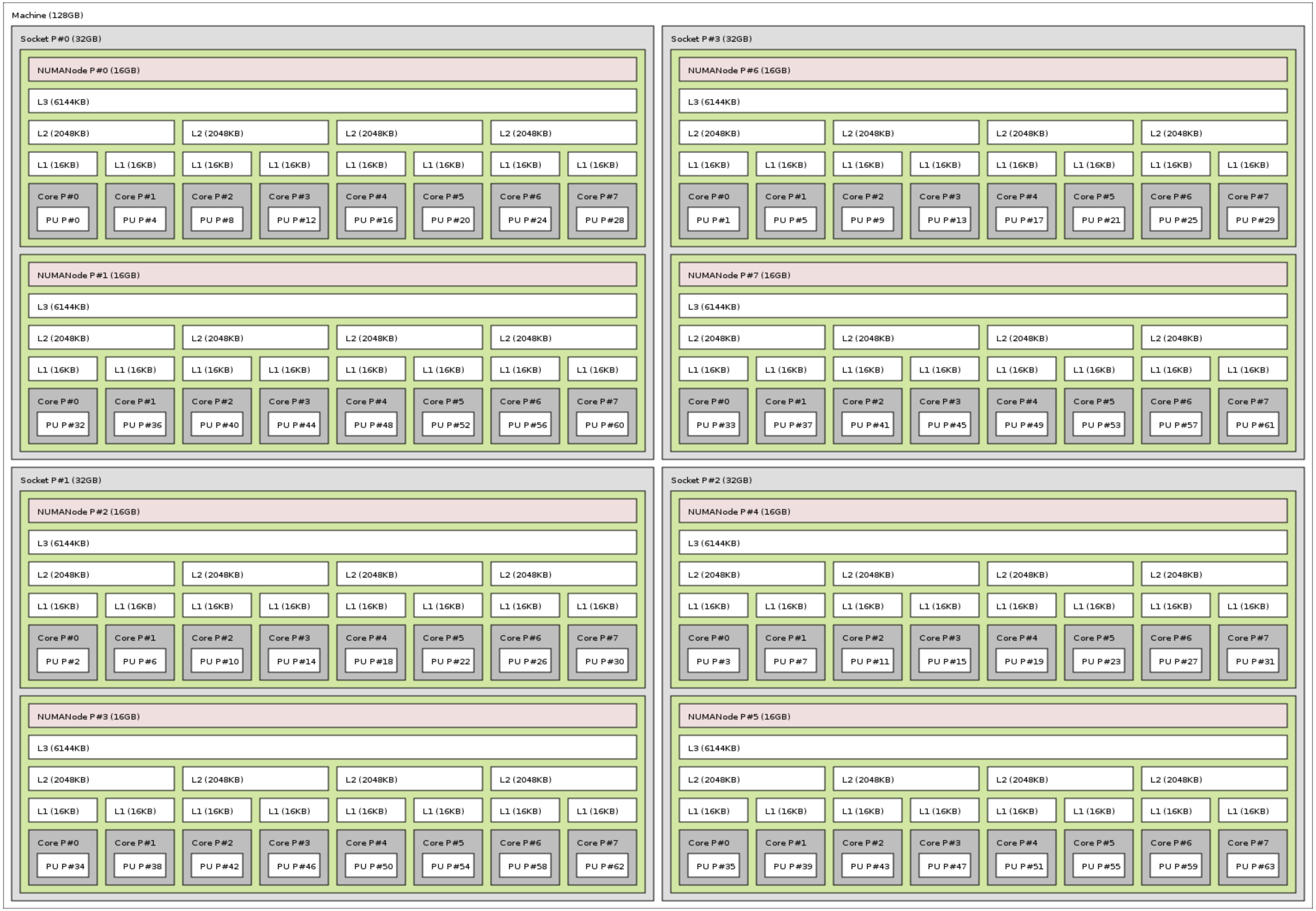
Example with MPI

- Let's say I have a 64-core AMD machine
  - Not unusual (about 6000$)
- I am running a MPI pingpong between pairs of cores
  - Open MPI 1.6
  - Intel MPI Benchmarks 3.2

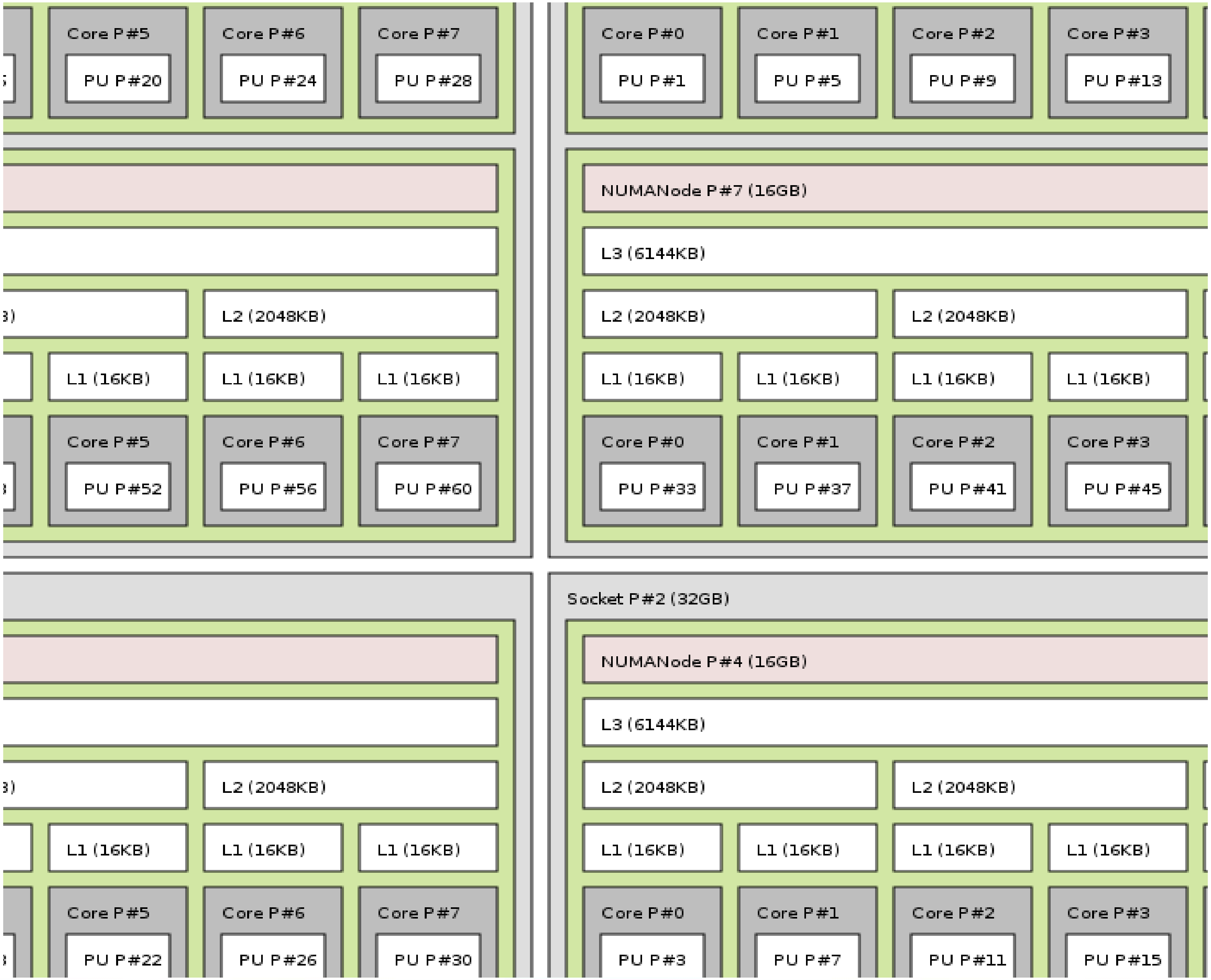
Example with MPI (2/3)

- Between cores 62 and 63
  - 1.39 µs latency – 1900MB/s throughput
- Between cores 60 and 63
  - 1.63 µs – 1400 MB/s – Interesting !
- Between cores 59 and 63
  - 0.68 µs – 3600 MB/s – What ?!
- Between cores 55 and 63
  - 1.24 µs – 2400 MB/s
- Between cores 31 and 63
  - 1.34 µs – 2100 MB/s

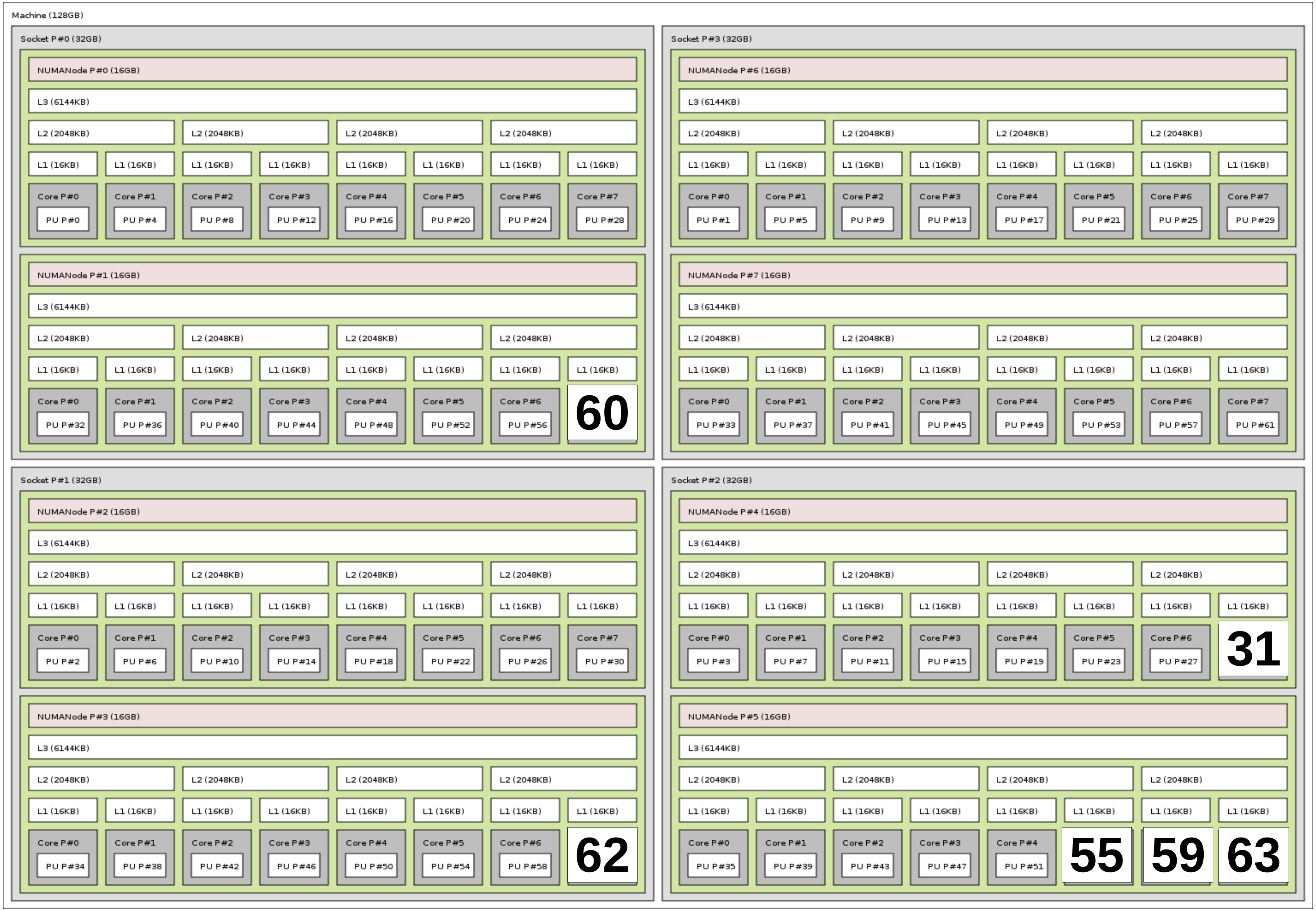
What is going on



What is going on (2/3)



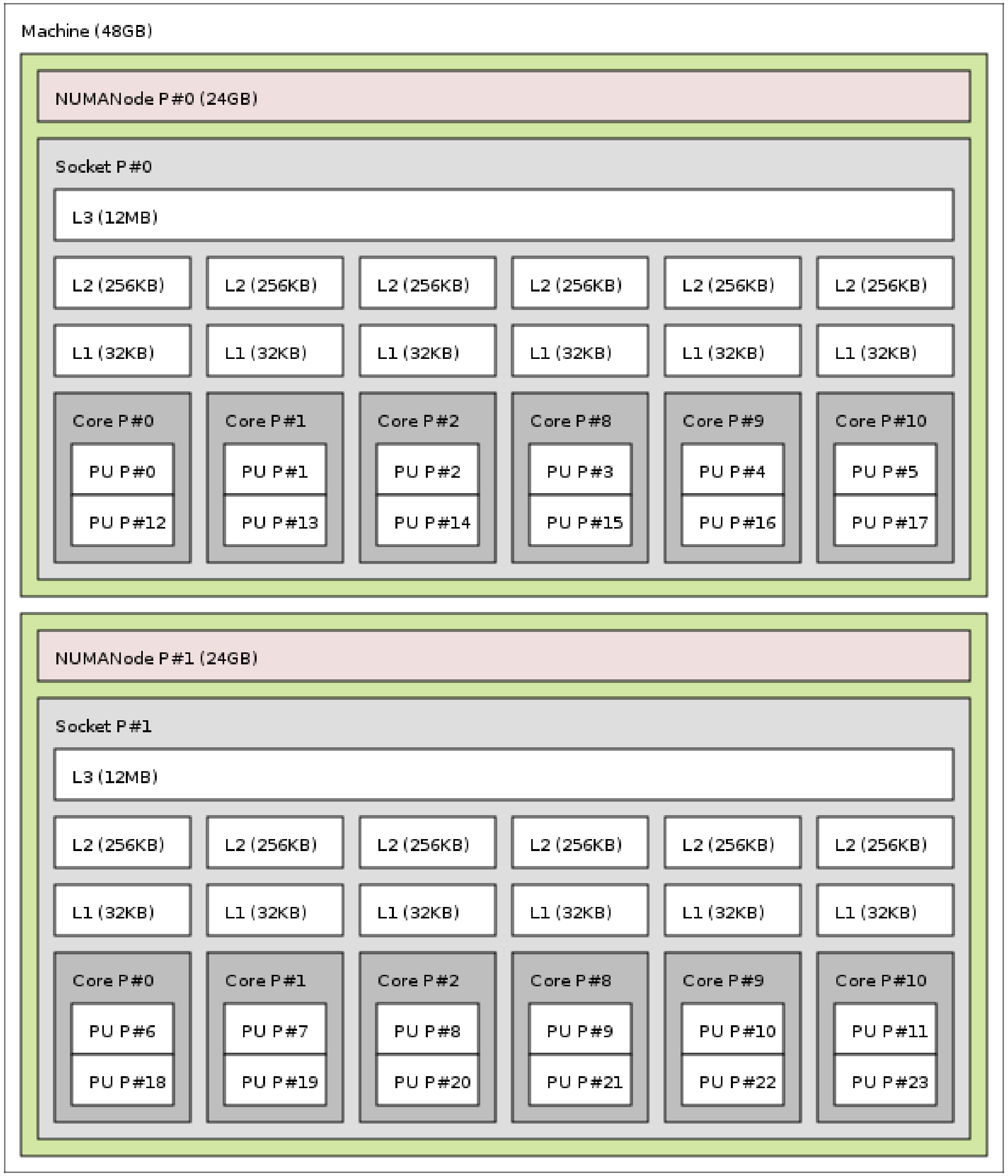
What is going on (3/3)



Example with MPI (3/3)

- Between cores that share a L2 cache
  - 0.68 µs – 3600 MB/s
- Between cores that only share a L3 cache
  - 1.24 µs – 2400 MB/s
- Between cores inside the same socket
  - 1.34 µs – 2100 MB/s
- Between cores of another socket
  - 1.39 µs – 1900MB/s
- Between cores of another socket further away
  - 1.63 µs – 1400 MB/s

Ok, what about Intel machines?



- Less hierarchy levels
  - 4 vs 3
  - HyperThreading?
- But same problems

First take away messages

- Locality matters to communication performance
  - Machines are really far from flat
- Cores/processors numbering is crazy
  - Never expect anything sane here

2 Bind your processes

Where does locality actually matter?

- MPI communication between processes on the same node
- Shared-memory too (threads, OpenMP, etc)
  - Synchronization
    - Barriers use caches and memory too
  - Concurrent access to shared buffers
    - Producer-consumer, etc
- 10 years ago, locality was mostly an issue for large NUMA SMP machines (SGI, etc)
  - Today it's **everywhere**
    - Because multicores and NUMA are everywhere

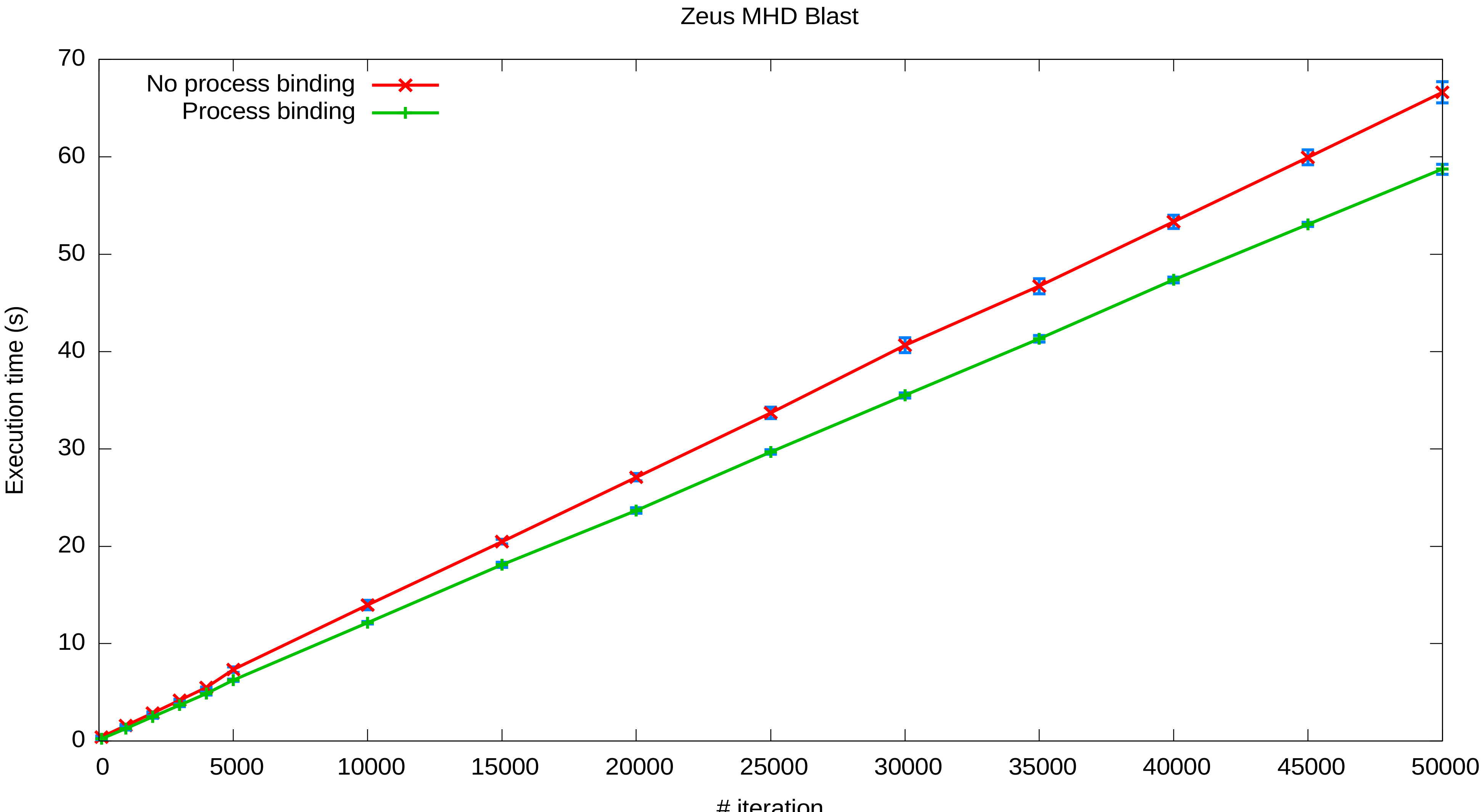
What to do about locality?

- Place processes/tasks according to their affinities
  - If two tasks communicate/synchronize/share a lot, keep them close
- Adapt your algorithms to the locality
  - Adapt communication/synchronization implementations to the topology
    - Ex: hierarchical barriers

Process binding

- Some MPI implementations bind processes by default (Intel MPI)
  - Because it's better for reproducibility
- Some don't (Open MPI, MPICH)
  - Because it may hurt your application
    - Oversubscribing?
- Binding doesn't guarantee that your processes are optimally placed
  - It just means your process won't move
    - No migration, less cache issues, etc

To bind or not to bind ?

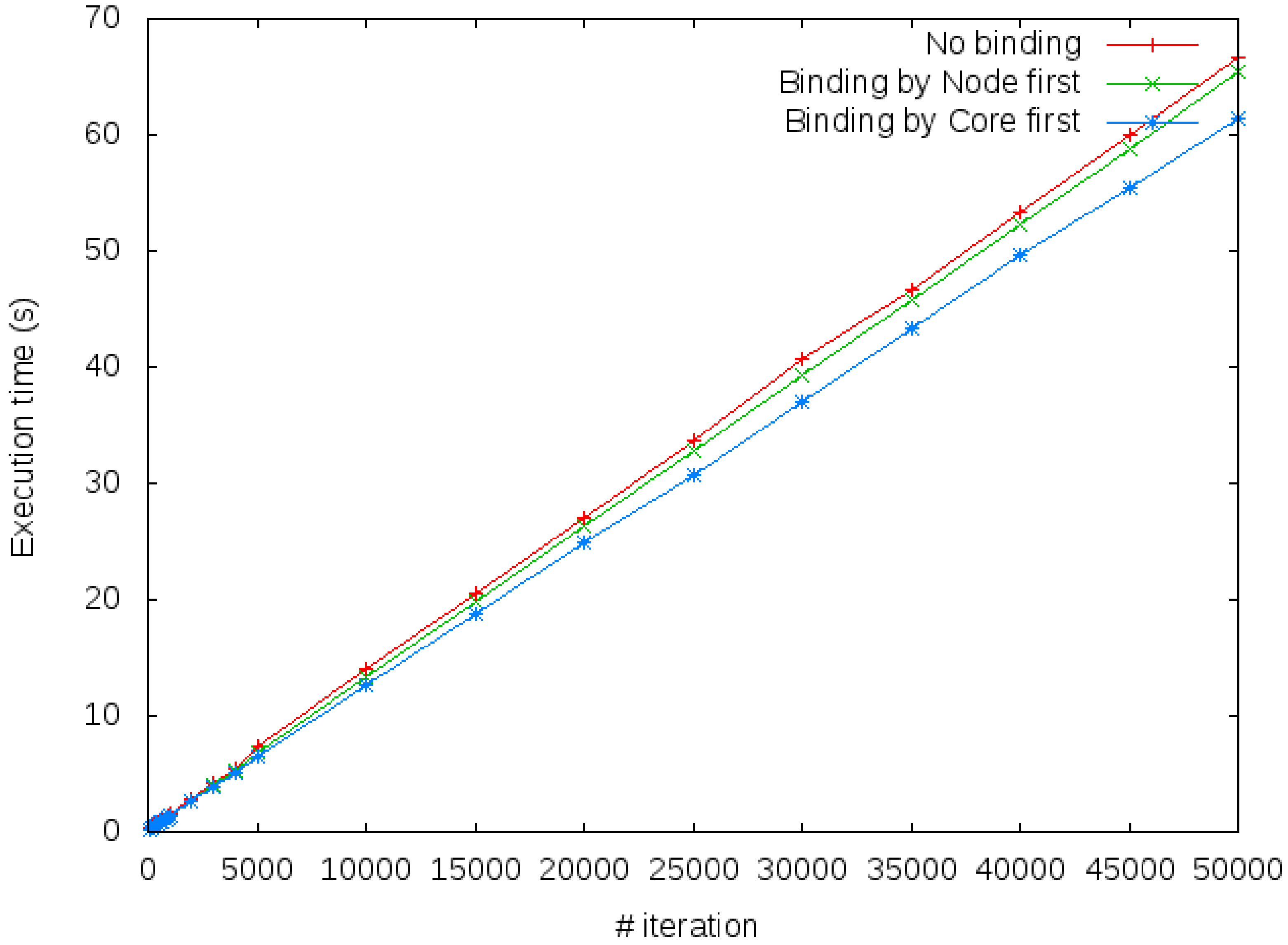


Zeus MHD Blast. 64 Processes/Cores. Mvapich2 1.8. + ICC

Where to bind ?

- Default binding strategies ?
  - By core first :
    - One process per core on first node,
      then one process per on second node, …
  - By node first :
    - One process on first core of each node,
      then one process on second core on each node, …
- Your application likely prefers one to the other
  - Usually the first one
    - Because you often communicate with nearby ranks

Binding strategy impact



How to bind in MPI? (more later)

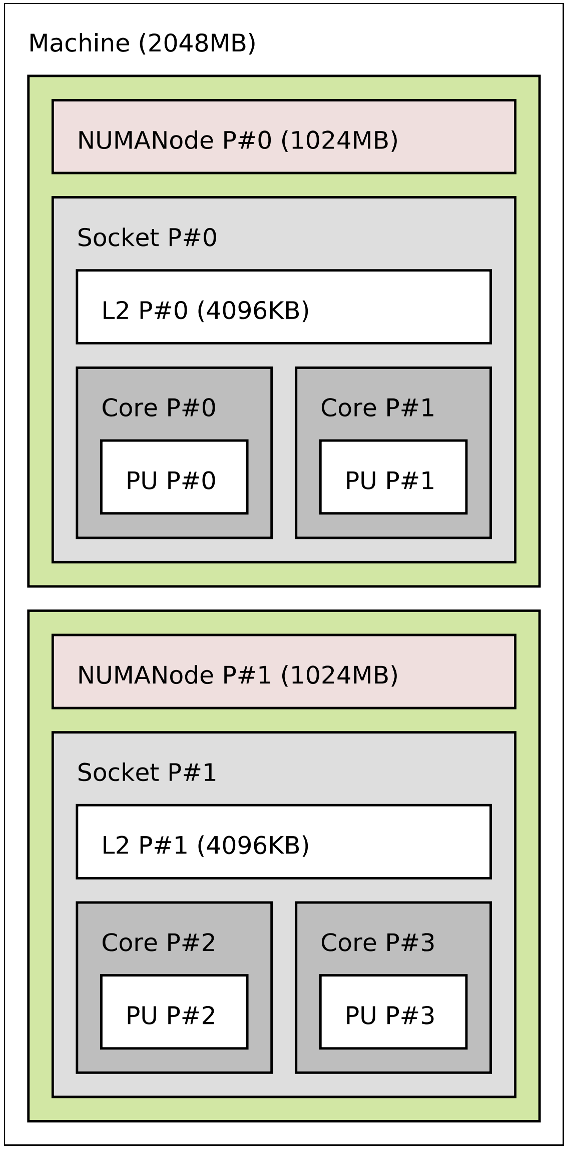
- MPI standard says nothing
- Open MPI
  - mpiexec --bind-to core -np 8 -H node1,node2 ./myprogram
- MPICH
  - mpiexec -bind-to core ...
- Manually
  - mpiexec
    -np 1 -H node1 numactl --physcpubind 0 ./myprogram :
    -np 1 -H node1 numactl --physcpubind 1 ./myprogram :
    -np 1 -H node2 numactl --physcpubind 0 ./myprogram
  - Rank files, etc

How to bind in OpenMP? (more later)

- Intel Compiler
  - KMP_AFFINITY=scatter or compact
- GCC
  - GOMP_CPU_AFFINITY=1,3,5,2,4,6

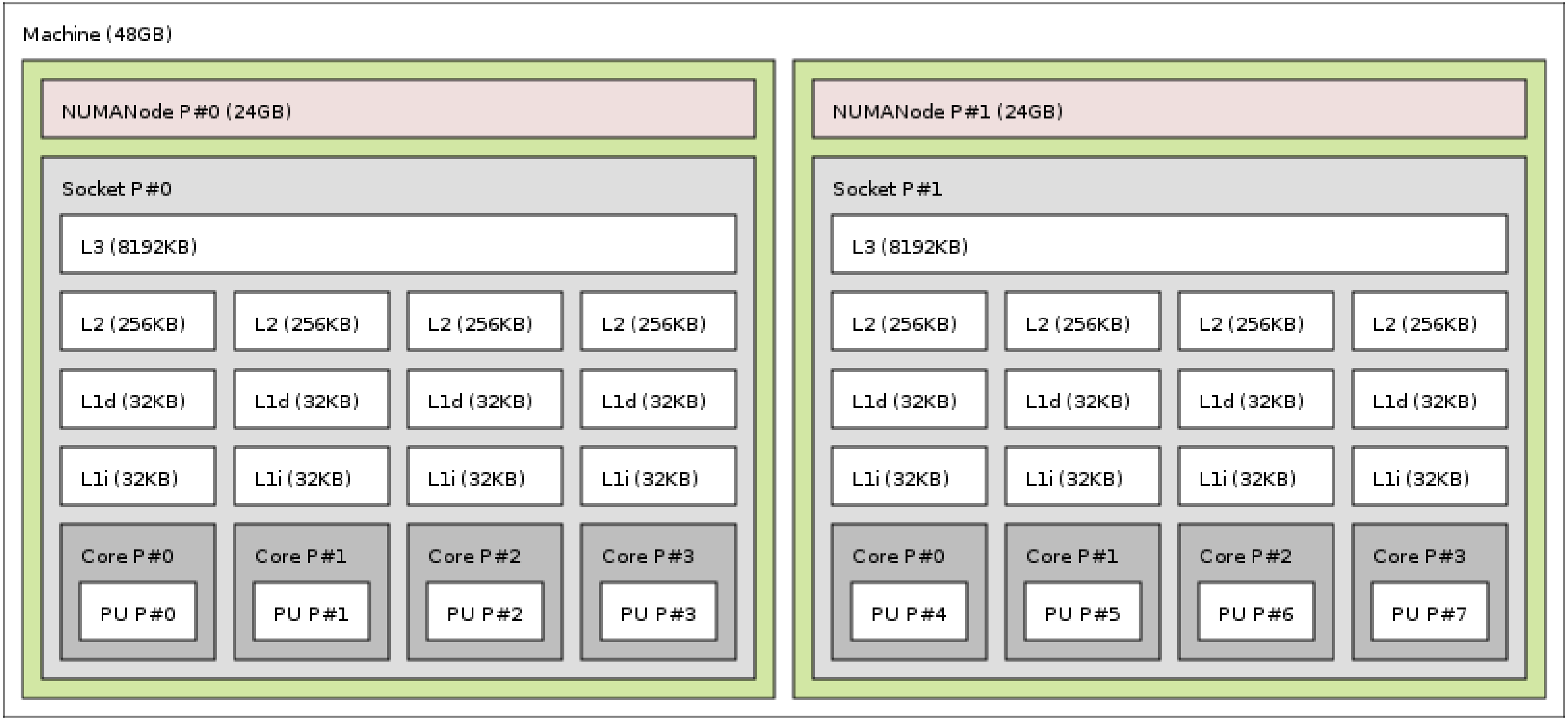
How do I choose?

- Dilemma
  - Use cores 0 & 1 to share cache and improve synchronization cost ?
  - Use core 0 & 2 to maximize memory bandwidth ?
- Depends on
  - The machine structure
  - The application needs

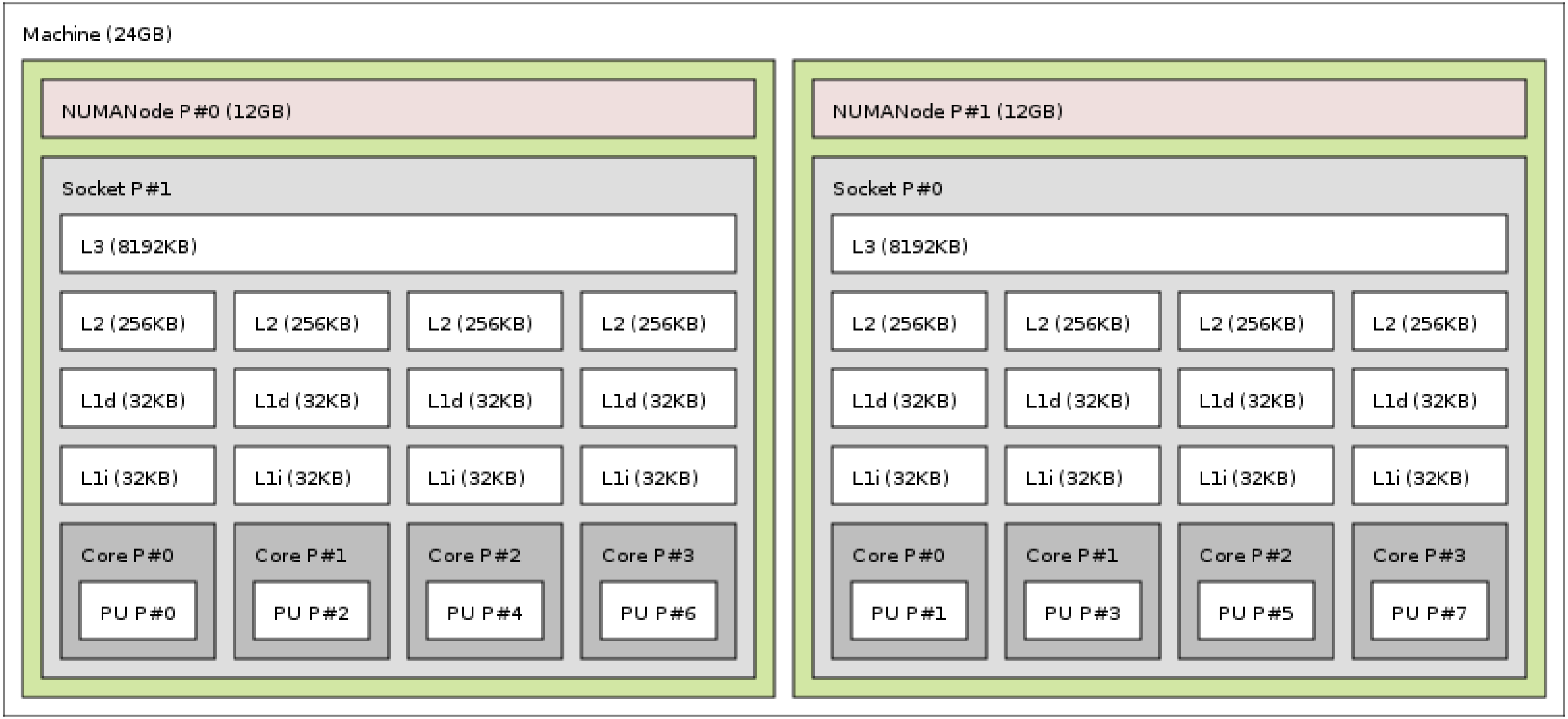


3 What's the actual problem ?

Example of dual Nehalem Xeon machine



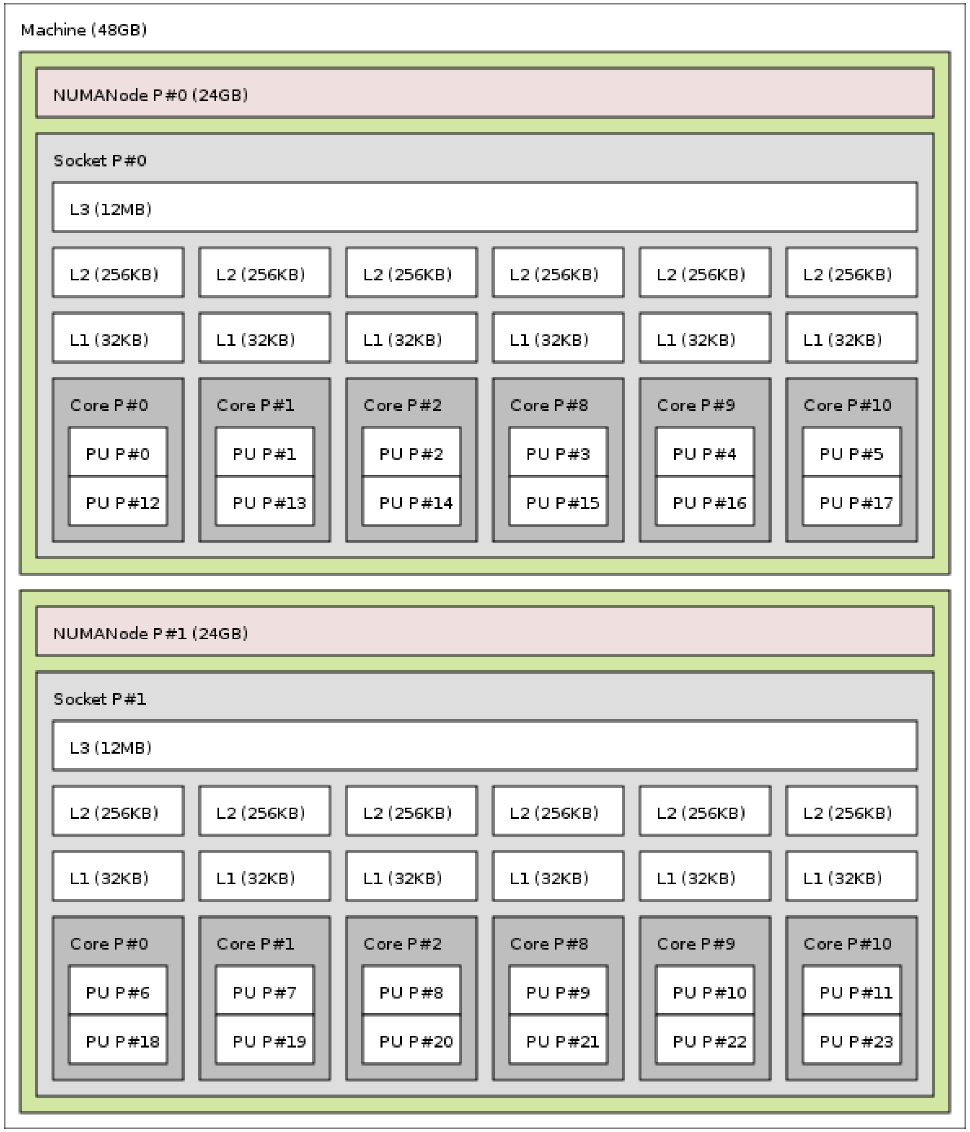
Another example of dual Nehalem Xeon machine



Processor and core numbers are crazy

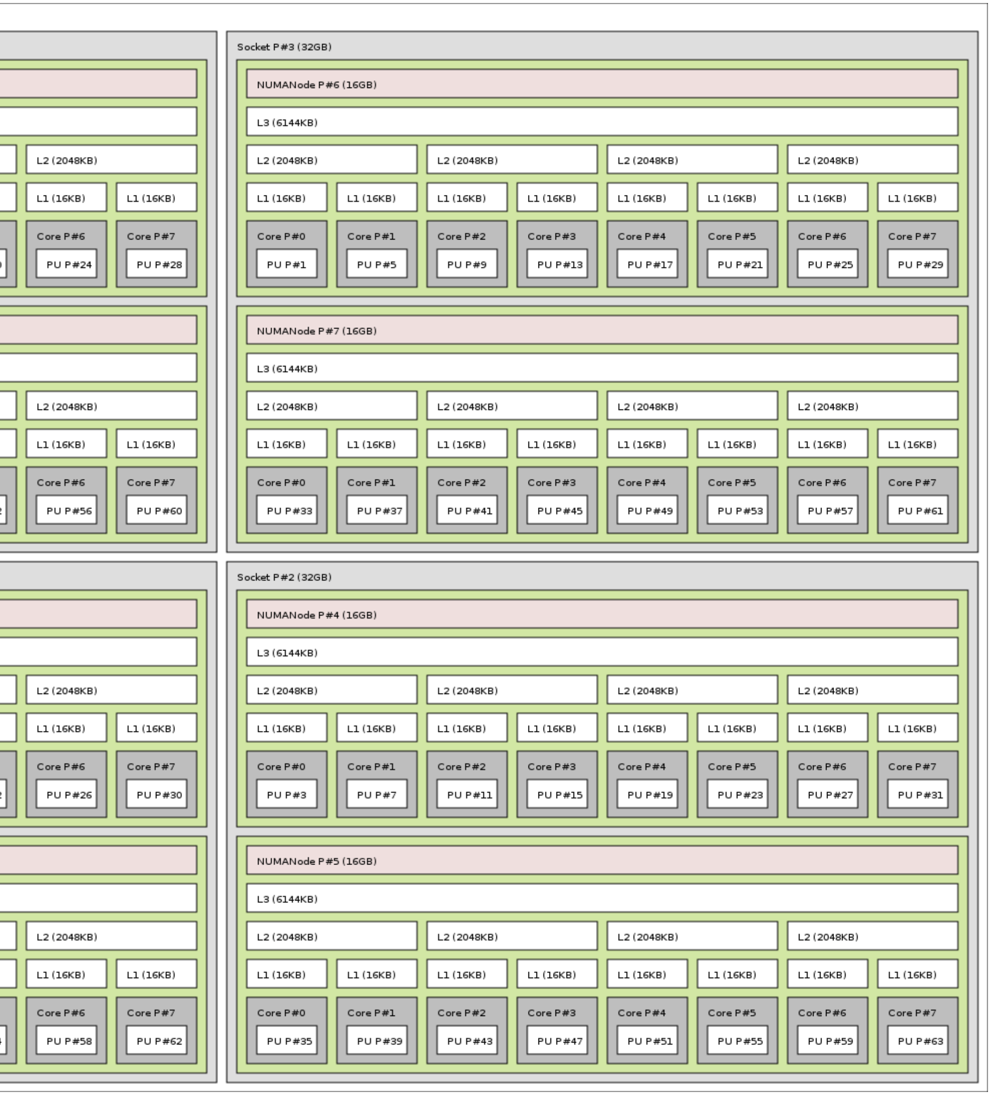
- Resources ordering is unpredictable
  - Ordered by any combination of NUMA/socket/core/hyperthread
  - Can change with the vendor, the BIOS version, etc
- Some resources may be unavailable
  - Batch schedulers can give only parts of machines
    - Core numbers may be non-consecutive, non starting at 0, etc
- Don't assume anything about indexes
  - Don't use these indexes
    - Or you won't be portable

Level ordering isn't much better



- Intel is usually
  - Machine
  - Socket = NUMA = L3
  - Core = L1 = L2
  - Hyperthread (PU)

Level ordering isn't much better (2/3)



- AMD is different
  - Machine
  - Socket
  - NUMA = L3
  - L2 = L1i
  - Core = L1d

Level ordering isn't much better (3/3)

- Sometimes there are multiple sockets per NUMA nodes
  - And different levels of caches
- Don't assume anything about level ordering
  - Or (again) you won't be portable
  - e.g.: Intel Compiler OpenMP binding may be wrong on AMD machines

Gathering topology information is difficult

- Lack of generic, uniform interface
  - Operating system specific
    - /proc and /sys on Linux
    - rset, sysctl, lgrp, kstat on others
  - Hardware specific
    - x86 cpuid instruction, device-tree, PCI config space, ...
- Evolving technology
  - AMD Bulldozer dual-core compute units
    - It's not two real cores, neither a dual-threaded core
  - New levels? New ordering?

Binding is difficult too

- Lack of generic, uniform interface, again
  - Process/thread binding
    - sched_setaffinity API changed twice on Linux
    - rset, ldom_bind, radset, affinity_set on others
  - Memory binding
    - mbind, migrate_pages, move_pages on Linux
    - rset, mmap, radset, nmadvise, affinity_set on others
  - Different constraints
    - Bind on single core only, on contiguous set of cores, on random sets ?
  - Many different policies

4

Introducing hwloc (Hardware Locality)

What hwloc is

- Detection of hardware resources
  - Processing units (PU), logical processors, hardware threads
    - Everything that can run a task
  - Memory nodes, shared caches
  - Cores, Sockets, … (things that contain multiple PUs)
  - I/O devices
    - PCI devices and corresponding software handles
- Described as a tree
  - Logical resource identification and organization
    - Based on locality

What hwloc is (2/2)

- API and tools to consult the topology
  - Which cores are near this memory node ?
  - Give me a single thread in this socket
  - Which memory node is near this GPU ?
  - What shared cache size between these cores ?
- Without caring about hardware strangeness
  - Non portable and crazy numbers, names, …
- A portable binding API
  - No more Linux sched_setaffinity API breakage
  - No more tens of different binding API with different types

What hwloc is not

- A placement algorithm
  - hwloc gives hardware information
  - You're the one that knows what your software does/needs
  - You're the one that must match software affinities to hardware localities
    - We give you the hardware information you need
- A profiling tool
  - Other tools (e.g. likwid) give you hardware performance counters
    - hwloc can match them with the actual resource organization

History

- Runtime Inria project in Bordeaux, France
  - Thread scheduling over NUMA machines (2003...)
    - Marcel threads, ForestGOMP OpenMP runtime
    - Portable detection of NUMA nodes, cores and threads
      - Linux wasn't that popular on NUMA platforms 10 years ago
        - Other Unixes have good NUMA support
    - Extended to caches, sockets, … (2007)
  - Raised questions for new topology users
    - MPI process placement (2008)

History

- Marcel's topology detection extracted as standalone library (2009)
- Noticed by the Open MPI community
  - They knew their PLPA library wasn't that good
- Merged both libraries as hwloc (2009)
- BSD-3
- Still mainly developed by Inria Bordeaux
  - Collaboration with Open MPI community
  - Contributions from MPICH, Redhat, IBM, Oracle, ...

Alternative software with advanced topology knowledge

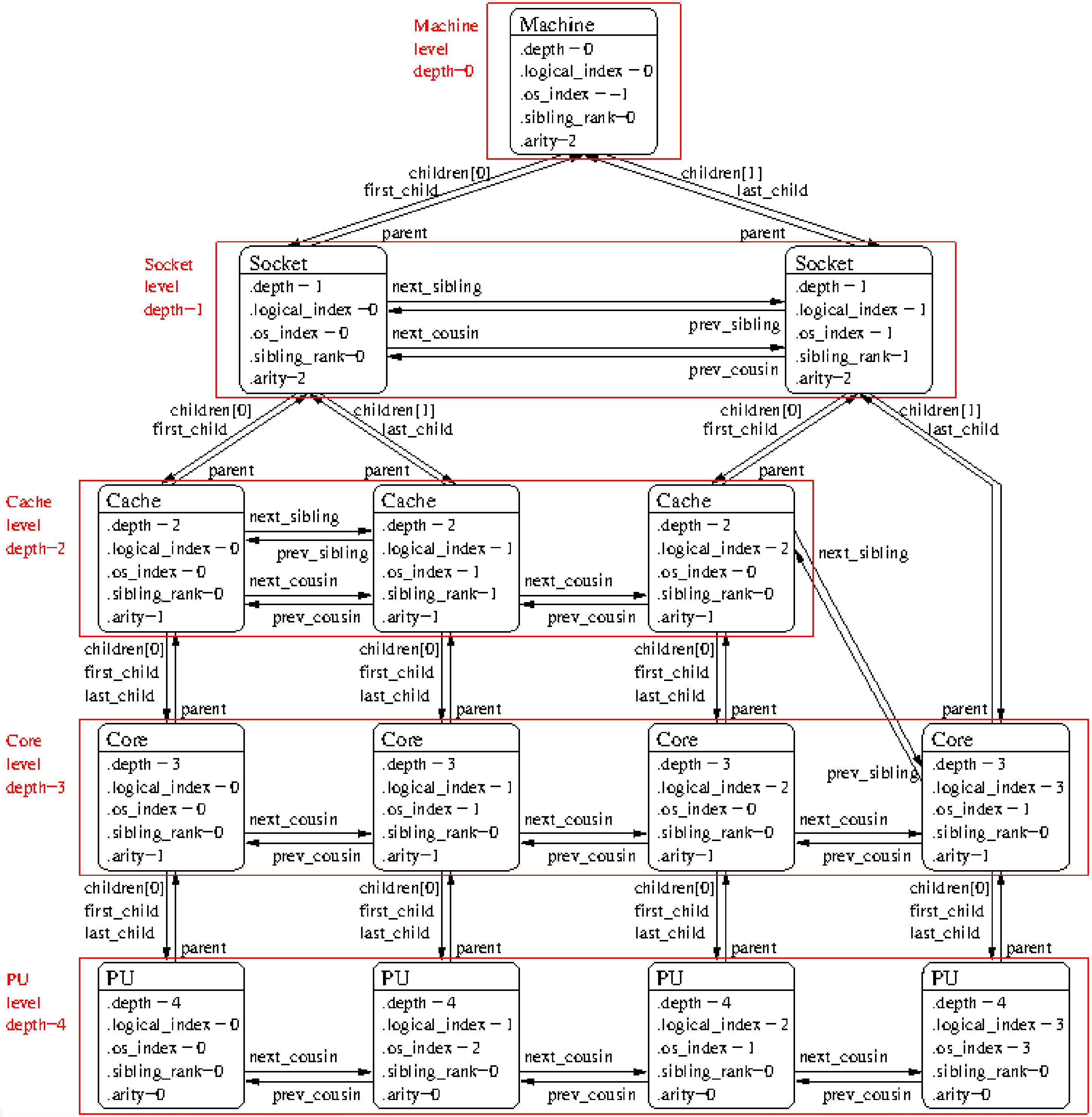
- PLPA (old Open MPI library)
  - Linux specific, no NUMA support, obsolete, dead
- libtopology (IBM)
  - Dead
- Likwid
  - x86 only, needs update for each new processor generation, no extensive C API
    - It's more kind of a performance optimization tool
- Intel Compiler (icc)
  - x86 specific, no API

Programming API

- Many hwloc command-line tools
- … but the actual hwloc power is in the C API
- Perl and Python bindings

hwloc's view of the hardware

- Tree of objects
  - Machines, NUMA memory nodes, sockets, caches, cores, threads
    - Logically ordered
  - Grouping similar objects using distances between them
    - Avoids enormous flat topologies
  - Many attributes
    - Memory node size
    - Cache type, size, line size, associativity
    - Physical ordering
    - Miscellaneous info, customizable


Machine level depth=0
Machine
.depth = 0
.logical_index = 0
.os_index = -1
.sibling_rank=0
.arity=2
children[0]
first_child
children[1]
last_child
parent
Socket level depth=1
Socket
.depth = 1
.logical_index =0
.os_index = 0
.sibling_rank=0
.arity=2
next_sibling
prev_sibling
next_cousin
prev_cousin
Socket
.depth = 1
.logical_index = 1
.os_index = 1
.sibling_rank=1
.arity=2
Cache level depth=2
Cache
.depth = 2
.logical_index = 0
.os_index = 0
.sibling_rank=0
.arity=1
Cache
.depth = 2
.logical_index = 1
.os_index = 1
.sibling_rank=1
.arity=1
Cache
.depth = 2
.logical_index = 2
.os_index = 0
.sibling_rank=0
.arity=1
Core level depth=3
Core
.depth = 3
.logical_index = 0
.os_index = 0
.sibling_rank=0
.arity=1
Core
.depth = 3
.logical_index = 1
.os_index = 1
.sibling_rank=0
.arity=1
Core
.depth = 3
.logical_index = 2
.os_index = 0
.sibling_rank=0
.arity=1
Core
.depth = 3
.logical_index = 3
.os_index = 1
.sibling_rank=0
.arity=1
PU level depth=4
PU
.depth = 4
.logical_index = 0
.os_index = 0
.sibling_rank=0
.arity=0
PU
.depth = 4
.logical_index = 1
.os_index = 2
.sibling_rank=0
.arity=0
PU
.depth = 4
.logical_index = 2
.os_index = 1
.sibling_rank=0
.arity=0
PU
.depth = 4
.logical_index = 3
.os_index = 3
.sibling_rank=0
.arity=0

Object information

- Type
- Optional name string
- Indexes (see later)
- Cpusets and Nodesets (see later)
- Tree pointers (*cousin, *sibling, arity, *child*, parent)
- Type-specific attribute union
  - obj->attr->cache.size
  - obj->attr->pcidev.linkspeed
- String info pairs

Physical or OS indexes

- obj->os_index
  - The ID given by the OS/hardware
- P#3
  - Default in lstopo graphic mode
  - lstopo -p
- NON PORTABLE
  - Depend on motherboards, BIOS, version, …
- DON'T USE THEM

Logical indexes

- obj->logical_index
  - The index among an entire level
- L#2
  - Default in lstopo except in graphic mode
  - lstopo -l
- Always represent proximity (depth-first walk)
- PORTABLE
  - Does not depend on OS/BIOS/weather
- That's what you want to use

But I still need OS indexes when binding ?!

- NO !
- Just use hwloc for binding, you won't need physical/OS indexes ever again

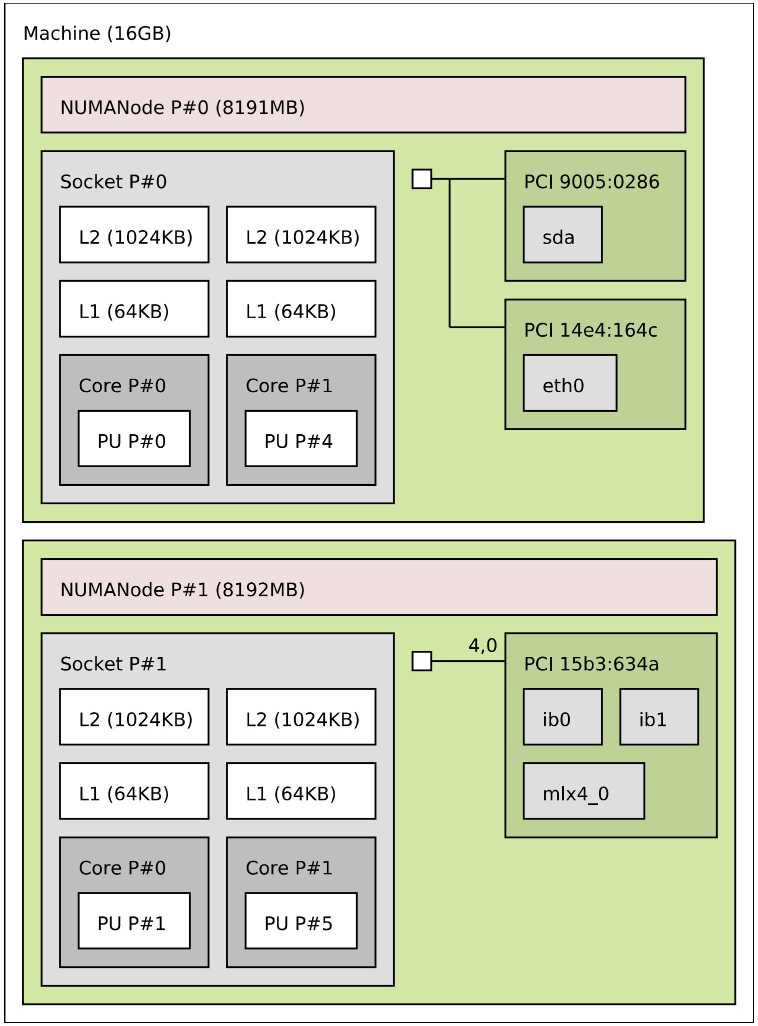
- If you want to bind the execution to a core
  - hwloc_set_cpubind(core->cpuset)
    - Other API functions for binding entire processes, single thread, memory, for allocating bound memory, etc.

Bitmap, CPU sets, Node sets

- Generic mask of bits : hwloc_bitmap_t
  - Possibly infinite
  - Opaque, used to describe object contents
    - Which PU are inside this object (obj->cpuset)
    - Which NUMA nodes are close to this object (obj->nodeset)
  - Can be combined to bind to multiple cores, etc.
    - and, or, xor, not, ...

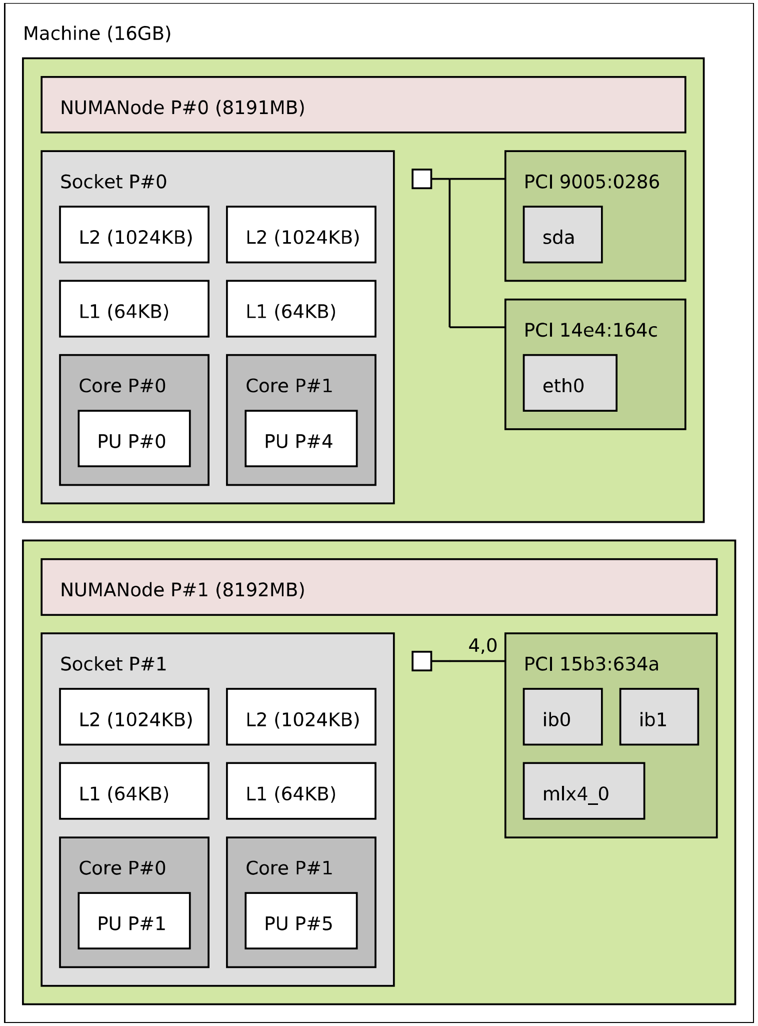
I/O devices

- Binding tasks near the devices they use improves their data transfer time
  - GPUs, high-performance NICs, InfiniBand, …
- You cannot bind tasks or memory on these devices
  - But these devices may have interesting attributes
    - Device type, GPU capabilities, embedded memory, link speed, …



I/O objects

- Some I/O trees are attached to the object they are close to
- PCI device objects
  - Optional I/O bridge objects
- How to match your software handle with a PCI device ?
  - OS/Software devices (when known)
    - sda, eth0, ib0, mlx4_0
- Disabled by default
  - Except in lstopo



Extended attributes

- obj->userdata pointer
  - Your application may store whatever it needs there
  - hwloc won't look at it, it doesn't know what's it contains
- (name,value) info attributes
  - Basic string annotations, hwloc adds some
    - HostName, Kernel Release, CPU Model, PCI Vendor, ...
  - You may add more

Configuring the topology

- Between hwloc_topology_init() and load()
  - hwloc_topology_set_xml(), set_synthetic()
  - hwloc_topology_set_flags(), set_pid()
  - hwloc_topology_ignore_type()

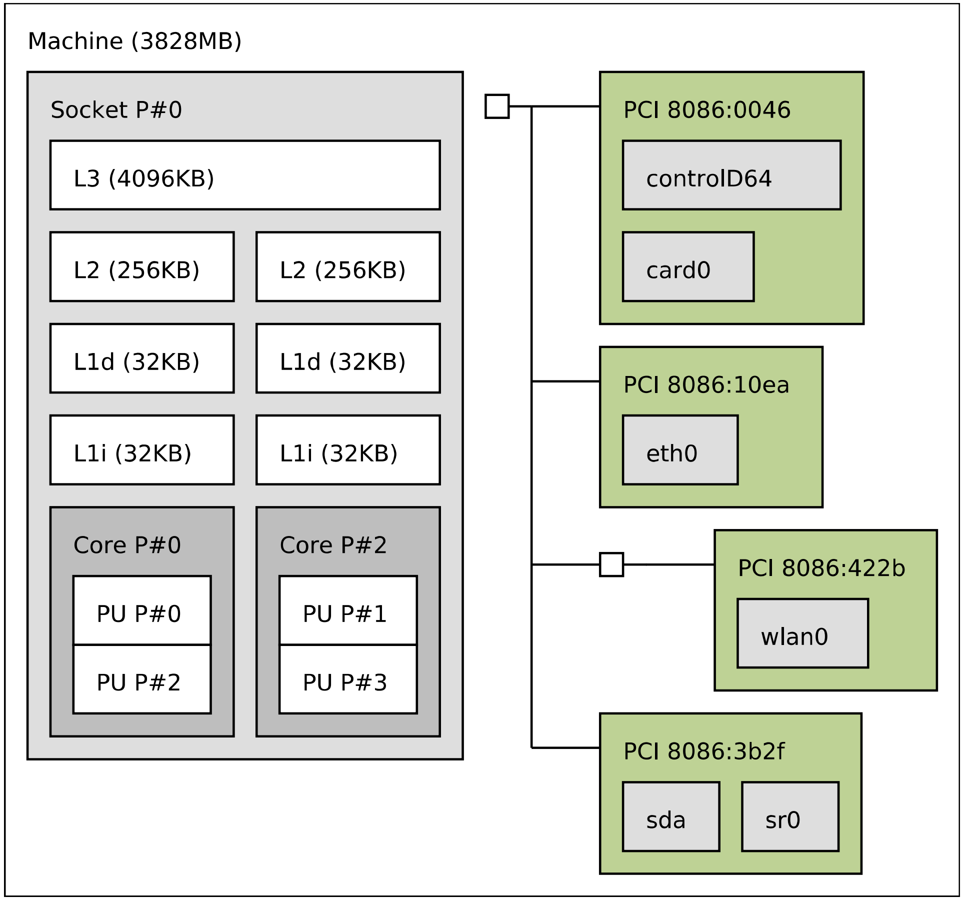
- After hwloc_topology_load()
  - hwloc_topology_restrict()
  - hwloc_topology_insert_misc_object...

Helpers

- hwloc/helpers.h contains a lot of helper functions
  - Iterators on levels, children, restricted levels
  - Finding caches
  - Converting between cpusets and nodesets
  - Finding I/O objects
  - And much more
- Use them to avoid rewriting basic functions
- Use them to understand how things work and write what you need

5 Command-line Tools

lstopo
(displaying topologies)




```
Machine (3828MB)
  Socket L#0 + L3 L#0 (4096KB)
    L2 L#0 (256KB) + Core L#0
      PU L#0 (P#0)
      PU L#1 (P#2)
    L2 L#1 (256KB) + Core L#1
      PU L#2 (P#1)
      PU L#3 (P#3)
  HostBridge L#0
    PCI 8086:0046
      GPU L#0 "controlD64"
    PCI 8086:10ea
      Net L#2 "eth0"
    PCIBridge
      PCI 8086:422b
        Net L#3 "wlan0"
    PCI 8086:3b2f
      Block L#4 "sda"
      Block L#5 "sr0"
```

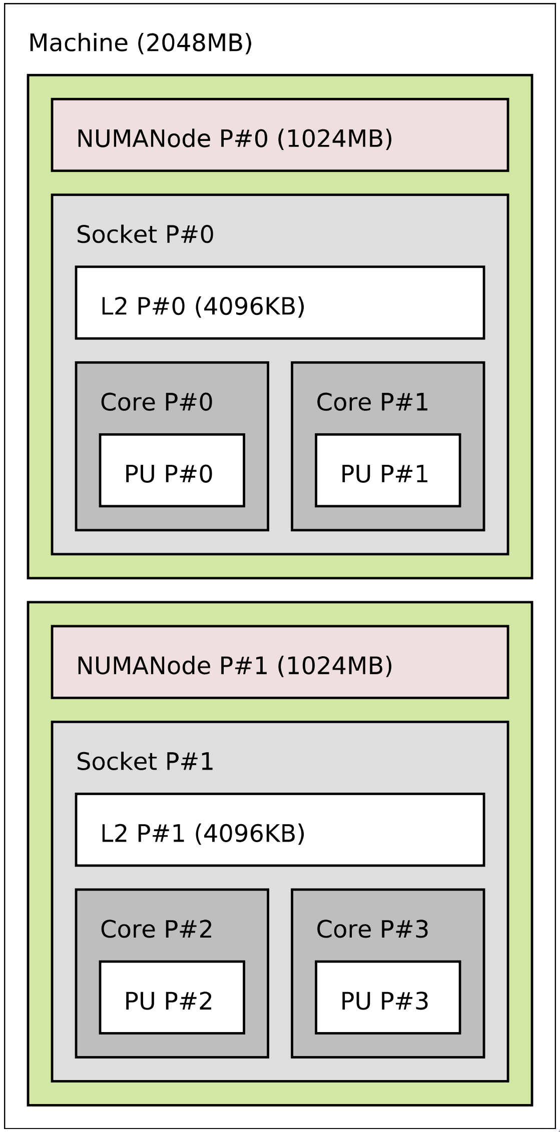
lstopo (2/2)

- Many output formats
  - Text, Cairo (PDF, PNG, SVG, PS), Xfig, Textual graphics (ncurses)
- XML dump
  - Save and quickly reload in another process
    - Instead of rediscovering everything again
  - Save for offline consultation
    - Batch schedulers placing processes on compute nodes
    - Remote debugging without access to the machine
- The output can be heavily tweaked
  - Useful for figures in your papers

hwloc-calc
(calculating with objects)

- Convert between ways to designate sets of CPUs, objects... and combine them

  $ hwloc-calc socket:1.core:1 ~pu:even
  0x00000008
  $ hwloc-calc --number-of core node:0
  2
  $ hwloc-calc --intersect pu socket:1
  2,3
- The result may be passed to other tools
- Multiple invocations may be combined
- I/O devices also supported
  $ hwloc-calc os=eth0
-



hwloc-bind
(binding processes, threads and memory)

- Bind a process to a given set of CPUs

  $ hwloc-bind socket:1 -- mycommand myargs...

  $ hwloc-bind os=mlx4_0 -- mympiprogram ...
- Bind an existing process

  $ hwloc-bind --pid 1234 node:0
- Bind memory

  $ hwloc-bind --membind node:1 --cpubind node:0 …
- Find out if a process is already bound

  $ hwloc-bind --get --pid 1234

  $ hwloc-ps

Other tools

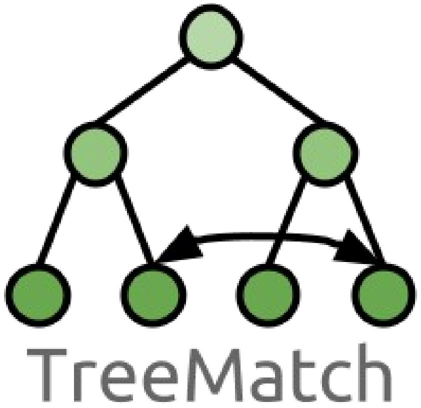
- Get some object information
  - hwloc-info (starting in hwloc v1.7)
- Generate bitmaps for distributing multiple processes on a topology
  - hwloc-distrib
- Save a Linux node topology info for debugging
  - hwloc-gather-topology
- More

6 Use cases

MPI process placement

- Given a matrix describing the communication pattern of an application
- How to place processes communicating intensively on nearby cores ?

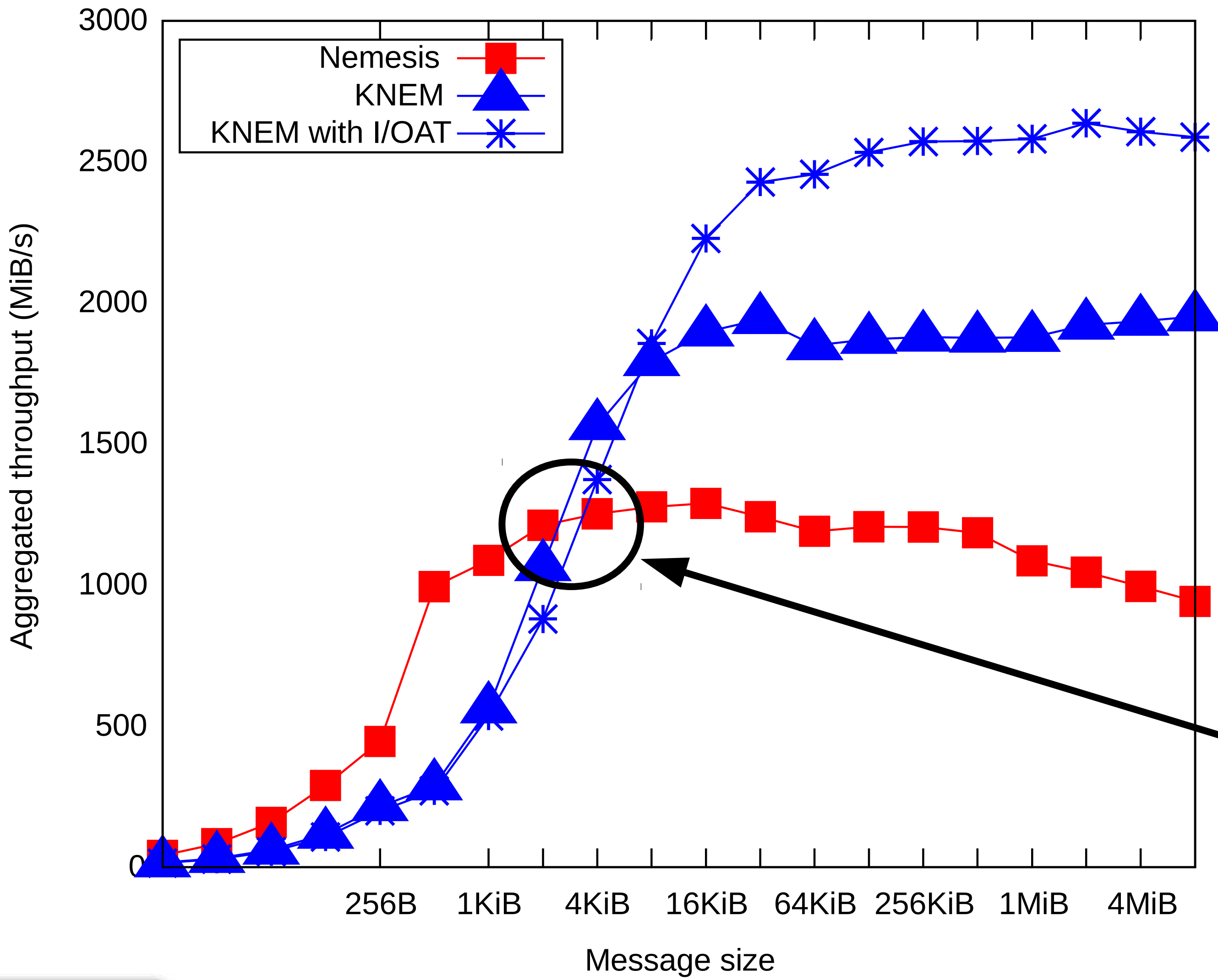
- This becomes a mapping of a tree of processes
  - Ordered by communication intensiveness
- … onto a tree of hardware resources
  - Given by hwloc



OpenMP thread scheduling with ForestGOMP

- OpenMP threads of the same parallel section often needs fast synchronization
  - Must stay together on the machine
    - Shared caches improve synchronization
- Build a tree of OpenMP teams and threads
  - Grouped by software affinities
- … and map it onto a tree of hardware caches, cores, NUMA nodes, …
  - Grouped by hardware locality

Topology-aware thresholds for MPI intra-node communication



Threshold that depends on shared cache size

Advanced binding strategies in MPI

- Open MPI
  - mpiexec --bind-to core --map-by core …
    - Map by node
  - mpiexec --bind-to core --mca rmaps_lama_map nsc
    - Map by node, then by socket, then by core
  - See mpiexec --help
- MPICH
  - mpiexec -bind-to core -map-by BSC ...
    - Map by node (board), then by socket, then by core
  - See mpiexec -bind-to -help

What about OpenMP ?

- Still far from MPI
  - Both for features and for portability of options
- Maybe more in OpenMP 4.0
  - We will see

7 Conclusion

More information

- The documentation
  - http://www.open-mpi.org/projects/hwloc/doc/
  - Related pages
    - http://www.open-mpi.org/projects/hwloc/doc/v1.6/pages.php
  - FAQ
    - http://www.open-mpi.org/projects/hwloc/doc/v1.6/a00014.php
- 3-4 hours tutorials with exercises on the webpage
- README and HACKING in the source
- hwloc-users@open-mpi.org for questions
- hwloc-devel@open-mpi.org for contributing
- hwloc-announce@open-mpi.org for new releases
- https://svn.open-mpi.org/trac/hwloc/report for reporting bugs

Thanks!

Questions?

Brice.Goglin@inria.fr