PATC
2016/06/06
Maison de la Simulation

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

Brice Goglin – TADaaM Team – Inria Bordeaux Sud-Ouest

Agenda

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

Materials

- All hwloc tutorials are available at

http://www.open-mpi.org/projects/hwloc/tutorials

1

Quick example as an Introduction

Machines are increasingly complex

Machines are increasingly complex

- Multiple processors
- Multicore processors (package = socket)
- Simultaneous multithreading
- Shared caches
- NUMA nodes
- Multiple GPUs, NICs, …

- We cannot expect users to understand all this

Example with MPI

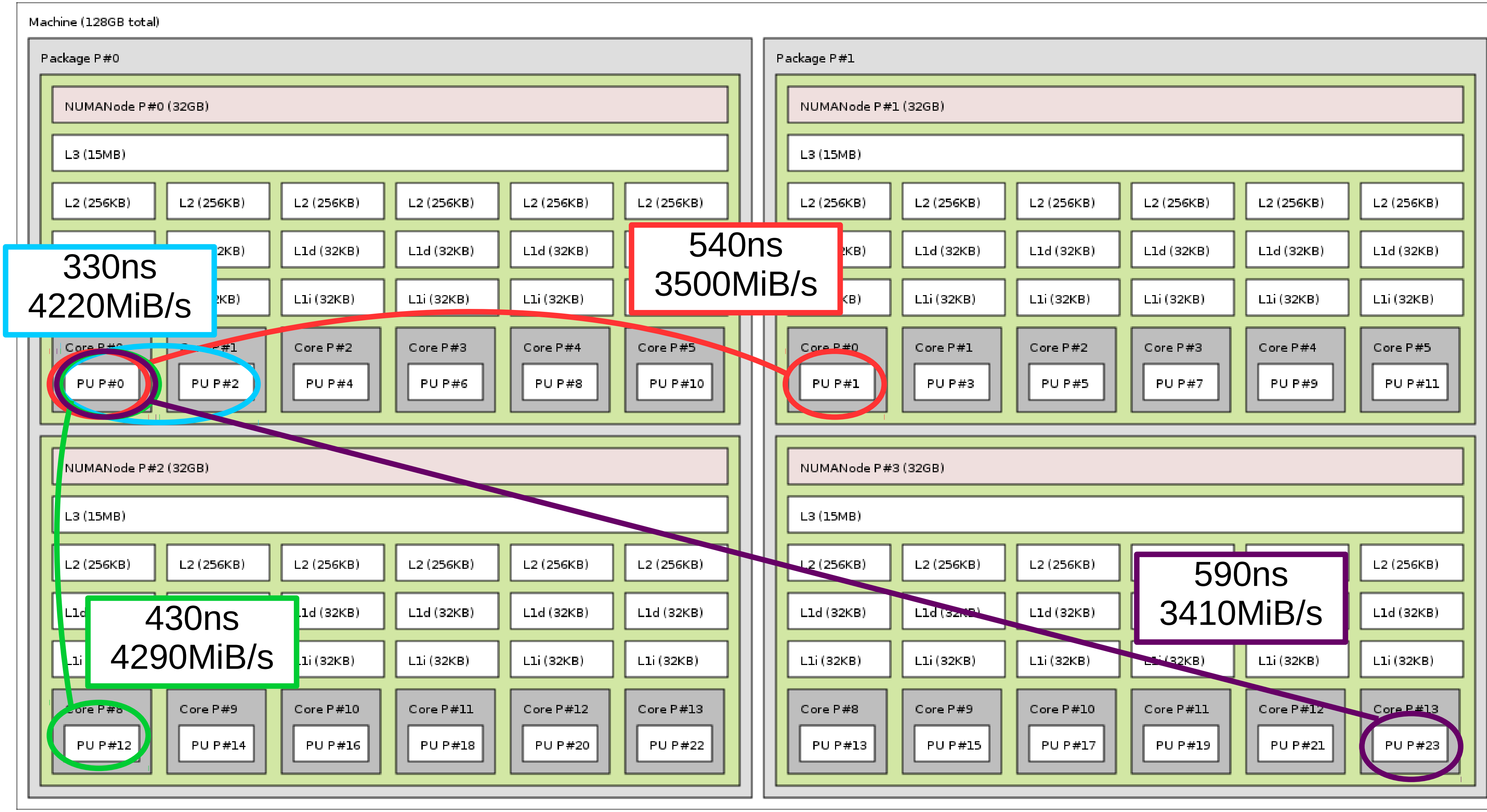
- Our latest cluster at Inria Bordeaux
  - 12-core Xeon E5-2600v3 with NVIDIA K40, etc.

- Nice, let's run some benchmarks!
  - Open MPI 1.8.1
  - Intel MPI benchmarks 3.2

Example with MPI – Results

- Between cores 0 and 1
  - 540ns, 3500MiB/s
- Between cores 0 and 2
  - 330ns, 4220MiB/s
- Between cores 0 and 12
  - 430ns, 4290MiB/s
- Between cores 0 and 23
  - 590ns, 3410MiB/s

What is going on?

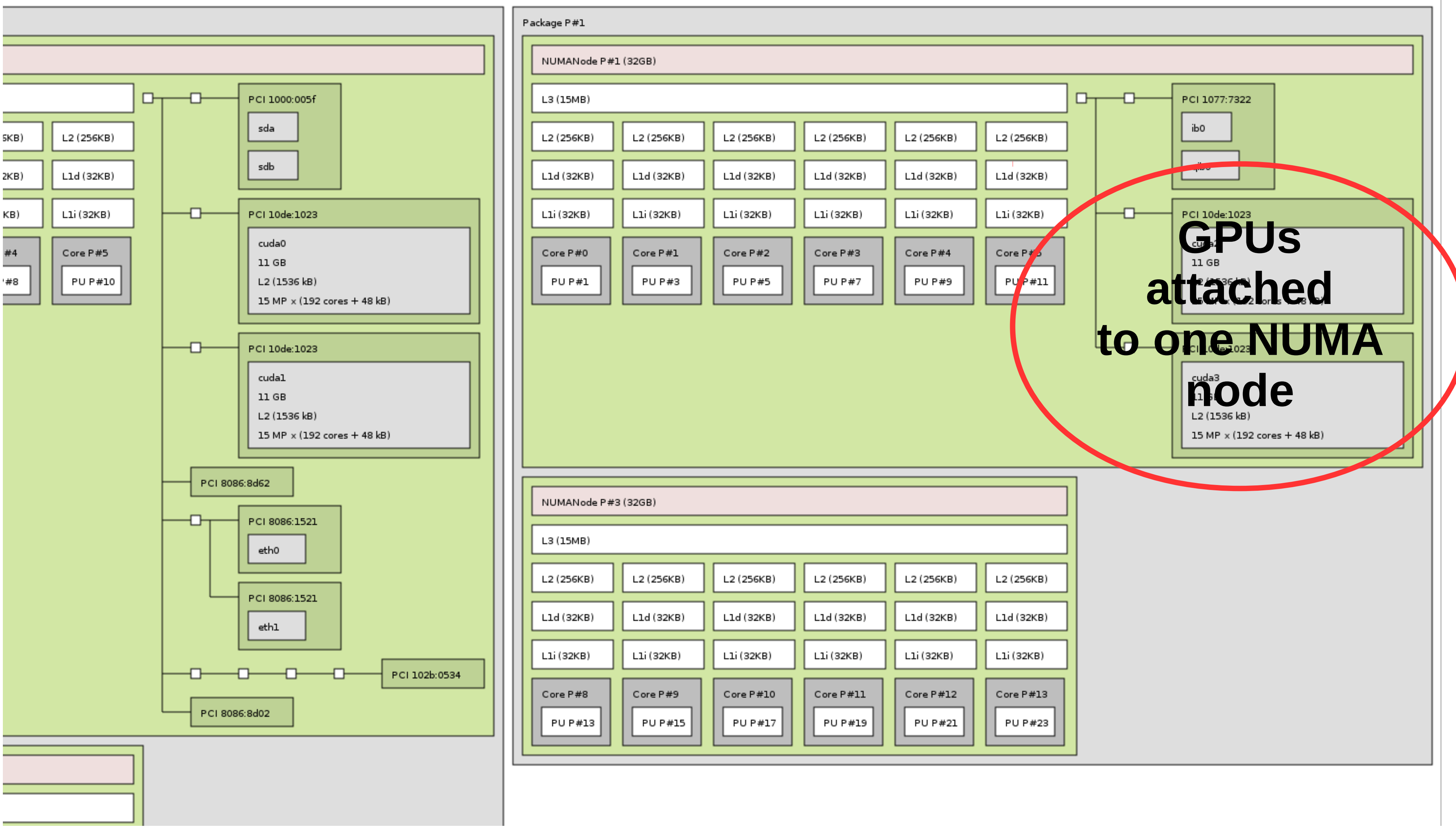


First take away messages

- Locality matters to communication performance
  - Machines are really far from flat
    - Similar issues with POWER8, AMD Opterons, Fujitsu Sparc XIfx

- Cores numbering is crazy
  - Never expect anything sane

It's actually worse than that



I/O affinity

- If you use GPUs or high performance networks, you must allocate host memory close to them
  - Otherwise DMA to GPUs slows down by 10-20% here
  - InfiniBand latency increases by 10%
- Need a way to know which cores/memory is close to which I/O device

2 Bind your processes

Where does locality actually matter?

- MPI communication performance varies with distance
  - Inside or outside nodes
- Shared memory too (threads, OpenMP, etc.)
  - Synchronization
    - Barriers use caches and memory too
  - Concurrent access to shared buffers
    - Producer-consumer, etc
- 15 years ago, locality was mostly an issue for large NUMA SMP machines (SGI, etc.)
  - Today it's everywhere
    - Because multicores and NUMA are everywhere

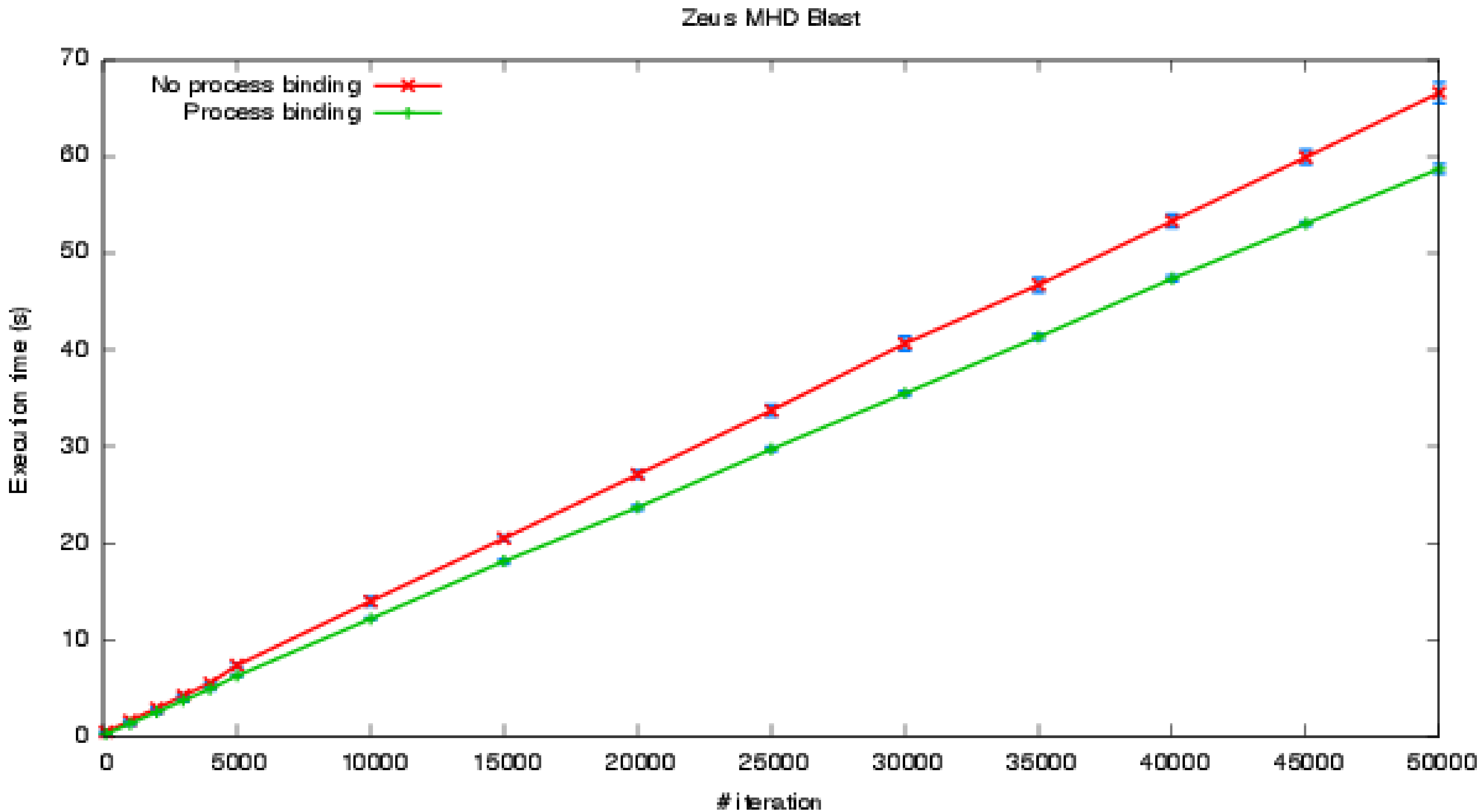
What to do about locality in runtimes?

- Place processes/tasks according to their affinities
  - If two tasks communicate/synchronize/share a lot, keep them physically close
    - Main focus of this talk
- Adapt your algorithms to the locality
  - Adapt communication/synchronization implementations to the topology
    - Hierarchical OpenMP barriers
    - Adapt your buffers to (shared) cache size

Process binding

- Some MPI implementations bind processes by default (Intel MPI, Open MPI 1.8 in some cases)
  - Because it's better for reproducibility
- Some don't
  - Because it may hurt your application
    - Oversubscribing? Dynamic processes?
- Binding doesn't guarantee that your processes are optimally placed
  - It just means your processes won't move
    - No migration, less cache issues, etc.

To bind or not to bind?

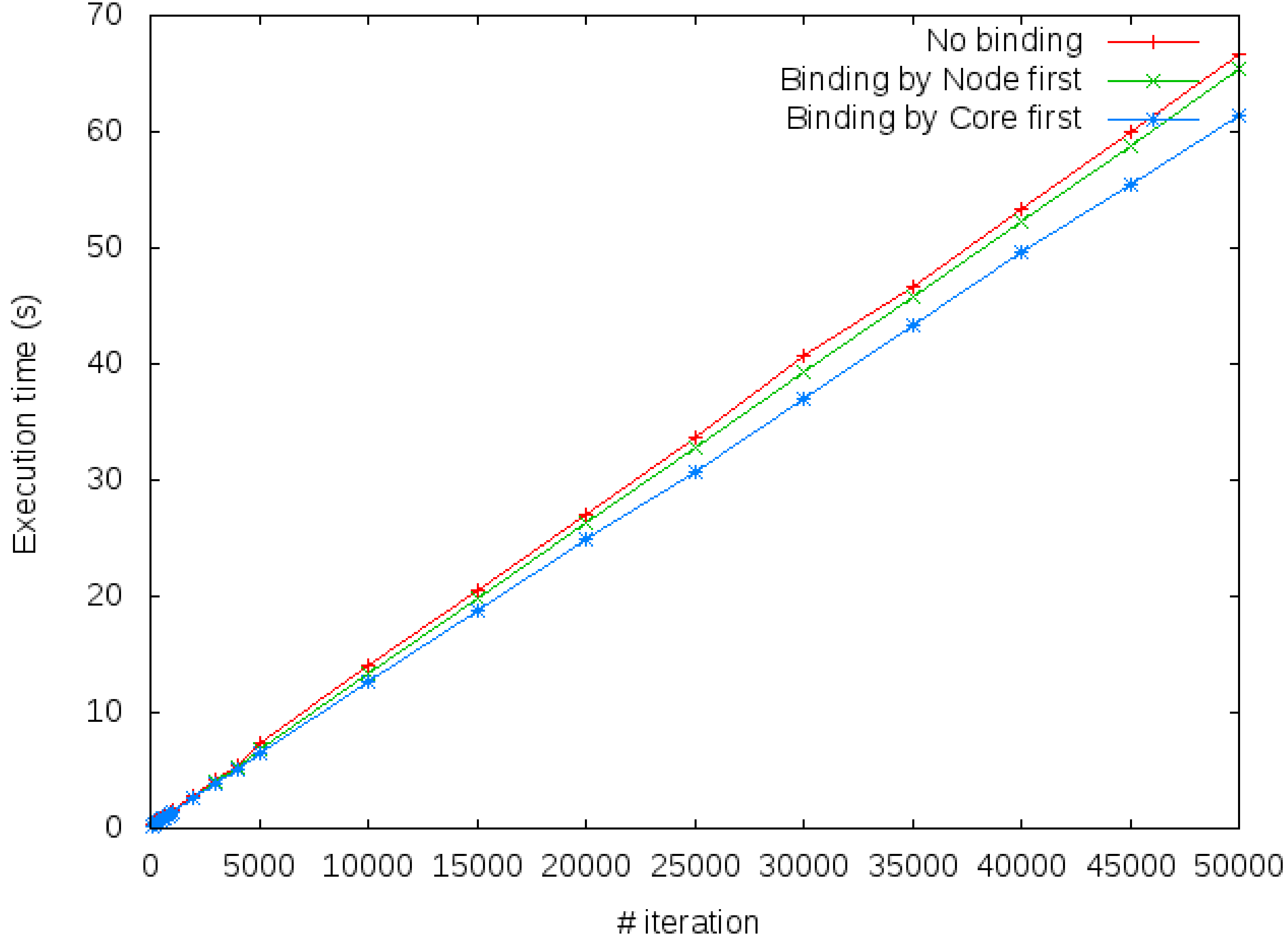


Zeus MHD Blast. 64 Processes/Cores. Mvapich2 1.8. + ICC

Ok, I need to bind.
But where?

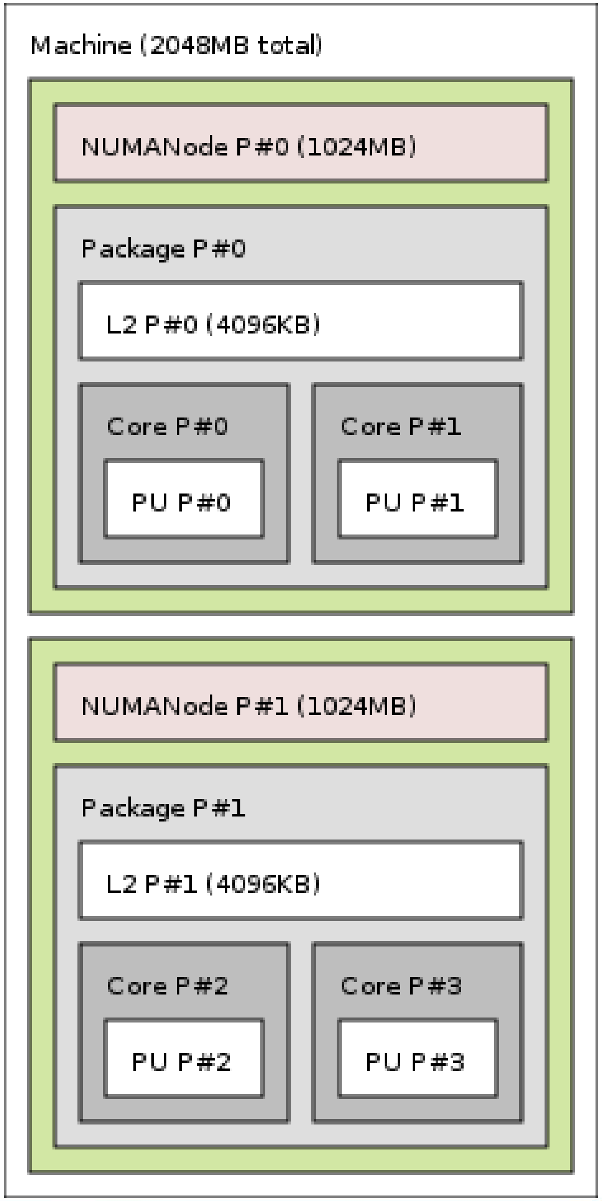
- Default binding strategies?
  - By core first (compact, --map-by core, etc.)
    - One process per core on first node,
      then one process per core on second node, …
  - By node first (scatter, --map-by node/socket, etc.)
    - One process on first core of each node,
      then one process on second core of each node, …
- Your application likely prefers one to the other
  - The first one?
    - Because your algorithms often communicate more between immediate neighbors
  - Sometimes the other one...

Binding strategy impact



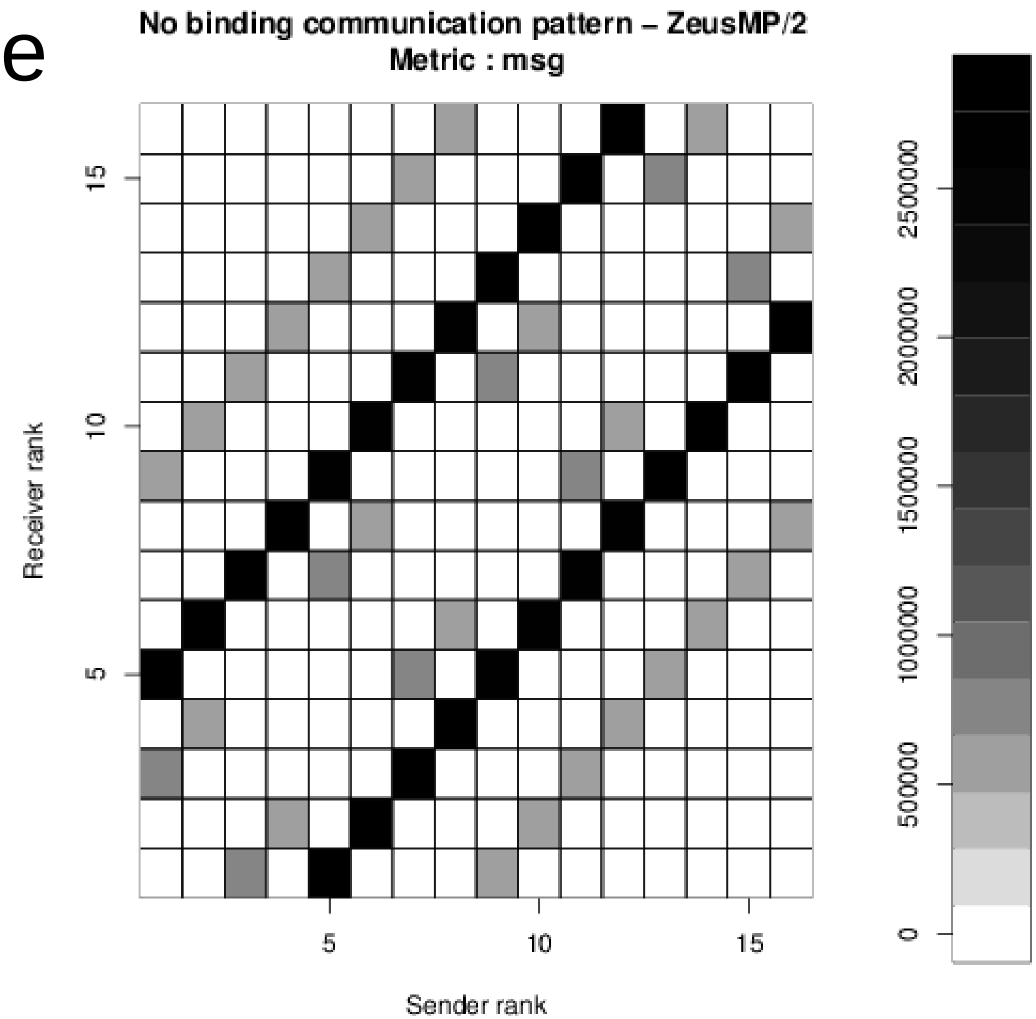
How do I choose?

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

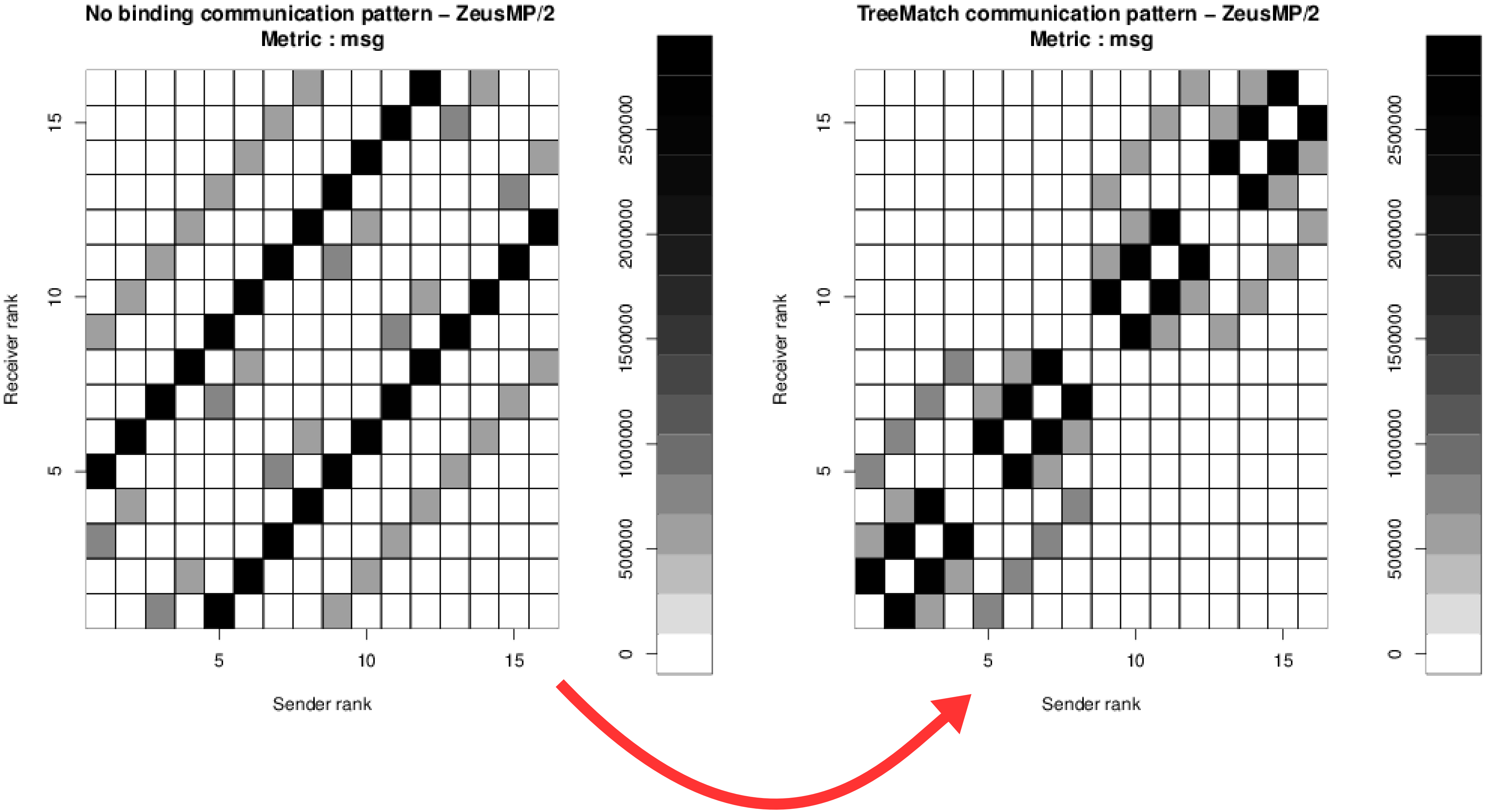


Topology-aware MPI process placement with TreeMatch

- Some tasks communicate a lot with each other
  - The physical distance will slow down some messages
  - Try to keep them close!
- Some don't
  - No constraint on placement



Reordering tasks to improve locality



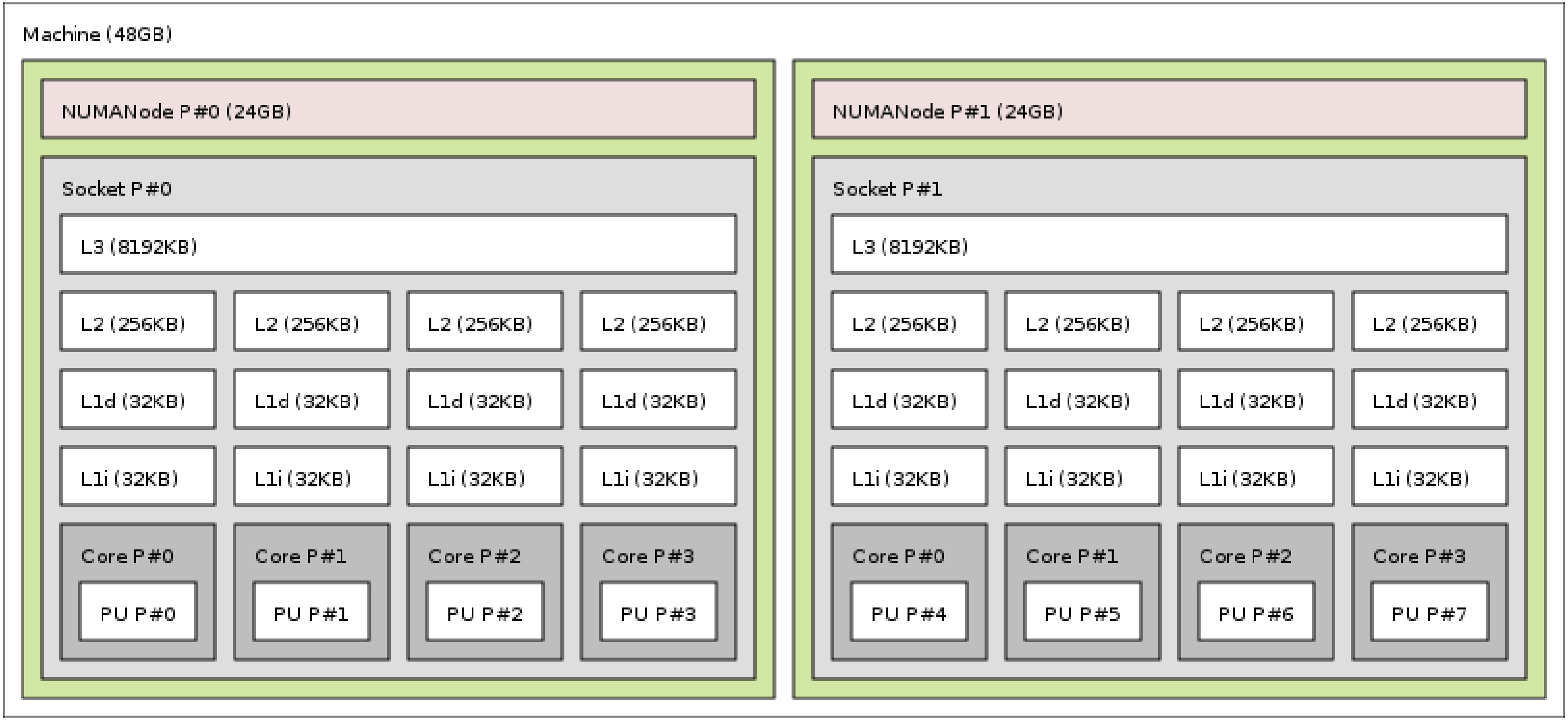
Reordering with TreeMatch

- At process launch-time
  - mpiexec options
- Dynamically
  - MPI_Dist_graph_create() to swap MPI ranks' roles between application steps
  - Charm++ load-balancer

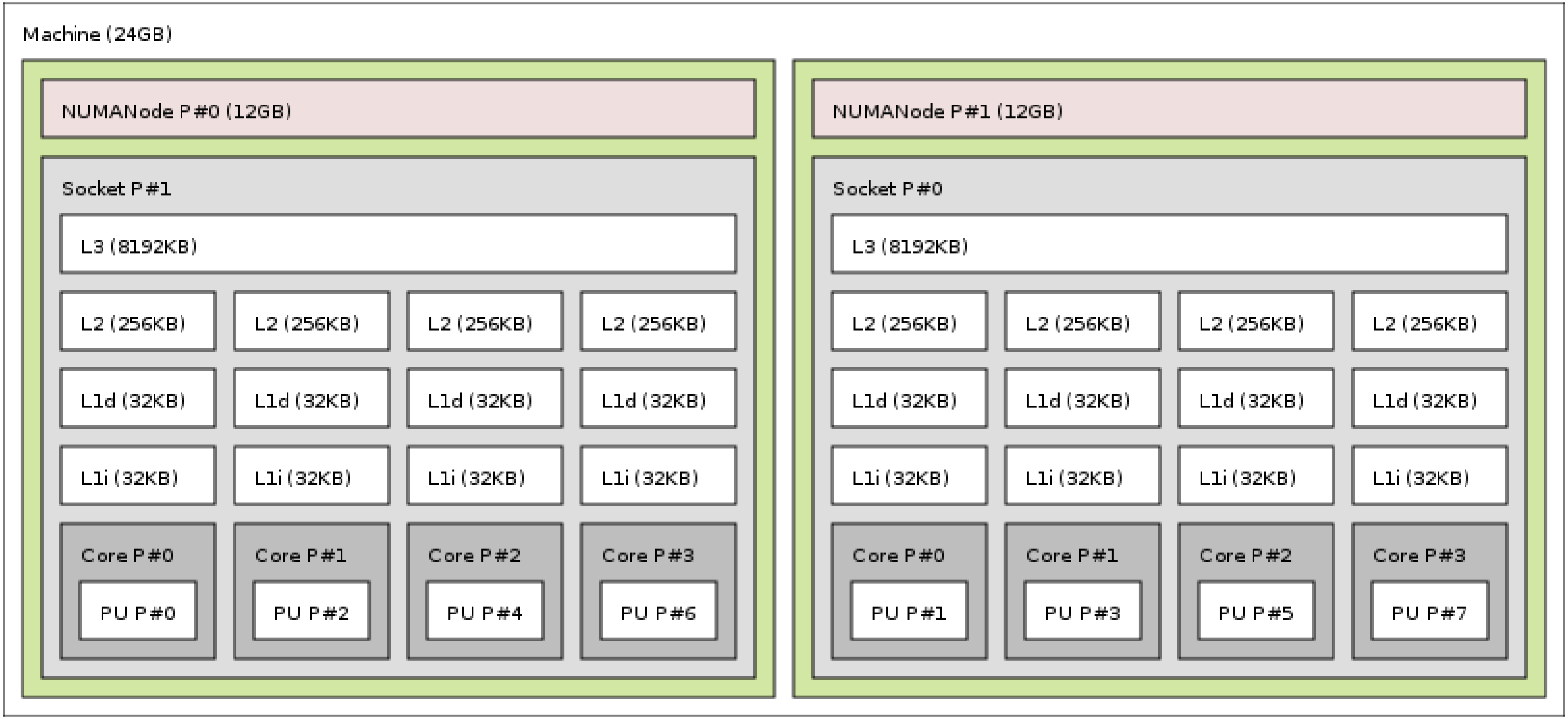
- The communication volume is unchanged
  - But big volumes move inside nodes
- Faster execution!

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

- Resource ordering/numbering is unpredictable
  - Can (and does) change with the vendor, BIOS version, etc.
- Some resources may be unavailable
  - Batch schedulers allocates parts of machines
    - Core numbers may be non-consecutive, not start at 0, etc.
- Don't assume anything about these numbers
  - **Otherwise your code won't be portable**

Vertical ordering of levels (who contains who)

Vertical ordering isn't reliable either

- Modern processors have 2 NUMA nodes each
  - Xeon E5v3, Opteron 6000, Power8, Sparc64 XIfx
  - But old platforms have multiple processor packages per NUMA nodes
- Levels of caches and sharing may vary
- Don't assume anything about vertical ordering
  - Or (again) your code won't be portable
  - e.g.: Even the Intel OpenMP binding isn't always correct

Gathering topology information is difficult

- Lack of generic, uniform interface
  - Operating system specific
    - /proc and /sys on Linux
    - rset, sysctl, lgrp, kstat on other OS
  - Hardware specific
    - x86 CPUID instruction, device-tree, PCI config space, etc.
- Evolving technology
  - AMD Bulldozer introduced dual-core Compute Units
    - It's not two real cores, neither one hyper-threaded core
  - New kinds of hierarchy/resources?
- And some BIOS report buggy information

Binding is difficult too

- Lack of generic, uniform interface (again)
  - Process/thread binding
    - sched_affinity() system call changed twice in Linux
  - Memory binding
    - 3 different system-calls on Linux
      - mbind(), migrate_pages(), move_pages()
  - Different constraints
    - Bind to single core only? To contiguous set of cores? To random sets of cores?
  - Many different policies

4 Introducing hwloc (Hardware Locality)

What hwloc is

- Detection of hardware resources
  - Processing units (PU) = logical processors, hardware threads, hyperthreads
    - Things that can run a task
  - Core, packages (sockets), … (things that contain PUs)
  - Memory nodes, shared caches
  - I/O devices
    - PCI devices and corresponding software handles
- Described as a tree
  - Logical resources identification and organization
    - **Based on locality**

What hwloc is (2/2)

- API and tools to consult the topology
  - Which cores are near this NUMA memory node ?
  - Give me a single thread in this package
  - Which NUMA 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 performance analysis tool

hwloc's History

- Ideas from Samuel Thibault's PhD on hierarchical thread scheduling (2003)
- Standalone library to ease MPI process placement (2009)
- Mainly developed by TADaaM@Inria Bordeaux
  - Within the Open MPI consortium
  - Collaboration with many industrial and academic partners
- BSD-3 license
- Many users

Alternative software with advanced topology knowledge

- numactl/libnuma
  - Only for NUMA + hardware threads
    - No cache, core, package/socket, etc.
- lscpu, lshw, lsusb, …
  - Specific to some resources
  - Inventory without locality information
- Likwid (performance optimization tool)
  - Now uses hwloc internally
- Intel Compiler (icc)
  - x86 specific, no API

hwloc Status

- Current stable release : 1.11.3 (April 2016)
- Support for most operating systems and HPC platforms

- Major release every 6 months
  - Backward compatible
- Next major release will be super-major
  - 2.0 expected in 2016H2
  - **Will break the ABI (not fully backward compatible)**
    - Fix bad ideas from the first hwloc API

hwloc's view of the hardware

- Tree of objects
  - Machines, NUMA memory nodes, packages, 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

Installing hwloc

- Packages available in Debian, Ubuntu, Redhat, Fedora, CentOS, ArchLinux, NetBSD, etc

- You want the development headers too
  - libhwloc-dev, hwloc-devel, …

Manual installation

- Take a recent tarball at http://www.open-mpi.org/projects/hwloc
- Dependencies
  - On Linux, numactl/libnuma development headers
  - Cairo headers for lstopo graphics
- ./configure --prefix=$PWD/install
  - Very few configure options
- Check the summary at the end of configure

Manual installation

- make
- make install
- Useful environment variables

export C_INCLUDE_PATH=$C_INCLUDE_PATH:<prefix>/include

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<prefix>/lib

export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:<prefix>/lib/pkgconfig

export PATH=$PATH:<prefix>/bin

export MANPATH=$MANPATH:<prefix>/share/man

Using hwloc for this tutorial

- Install hwloc on your preferred cluster
- And install it on your laptop too
  - It will make remote machine consulting easier

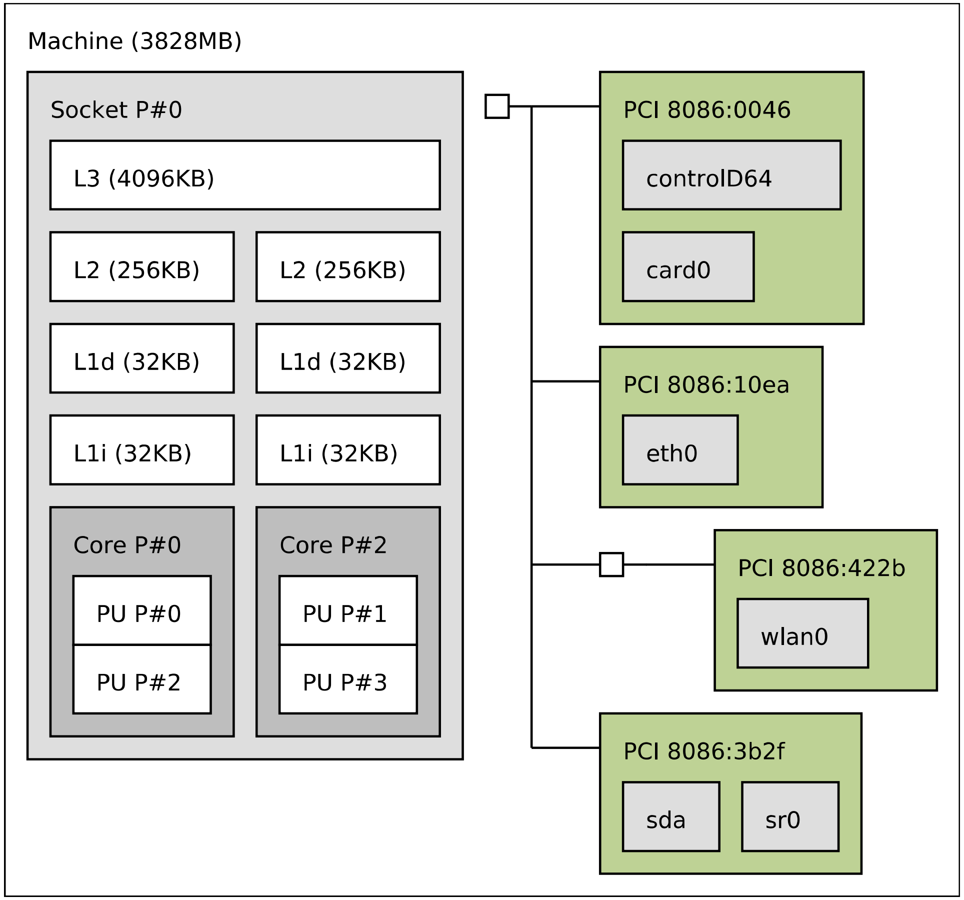
Using hwloc

- Many hwloc command-line tools
  - lstopo and hwloc-*
- … but the actual hwloc power is in the C API

- Perl and Python bindings

5 Command-line Tools

lstopo
(displaying topologies)




```
Machine (3828MB)
  Package 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

- Many output formats
  - Text, Cairo (PDF, PNG, SVG, PS), Xfig, ncurses
    - Automatically guessed from the file extension
- XML dump/reload
  - Faster, convenient for remote debugging
- Configuration options for nice figures for papers
  - Horizontal/Vertical placement
  - Legend
  - Ignoring some resources
  - Creating fake topologies

lstopo

```
$ lstopo
$ lstopo --no-io -
$ lstopo myfile.png
$ ssh host lstopo saved.xml
$ lstopo -i saved.xml
$ ssh myhost lstopo -.xml | lstopo --if xml -i -
$ lstopo -i “numa:4 package:2 core:2 pu:2”
```

hwloc-bind
(binding processes, threads and memory)

- Bind a process to a given set of CPUs

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

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

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

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

  $ hwloc-bind --get --pid 1234

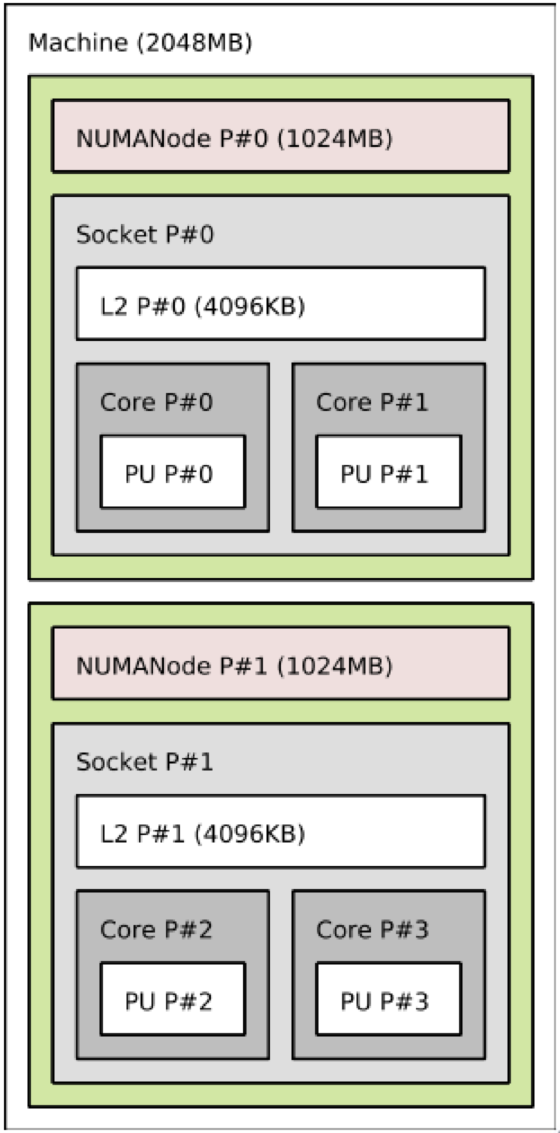
  $ hwloc-ps

hwloc-calc (calculating with objects)

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

  $ hwloc-calc package:1.core:1 ~pu:even
  0x00000008
  $ hwloc-calc --number-of core numa:0
  2
  $ hwloc-calc --intersect pu package: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



Other tools

- Get some object information
  - hwloc-info
- Generate bitmaps for distributing multiple processes on a topology
  - hwloc-distrib
- Save a Linux node topology info for debugging
  - hwloc-gather-topology
- Manipulating multiple topologies, etc.

Hands-on lstopo

- Gather the topology of one server
- Display it on another machine
- Hide caches
- Remove the legend
- Restrict the display to a single package
- Export to PDF

Hands-on hwloc-bind and hwloc-calc

- Bind a process to a core and verify its binding
- Find out how many cores are in the second NUMA node
- Find out the physical numbers of all non-first hyperthreads
- Find out which cores are close to InfiniBand
- Compare the DMA bandwidth from GPU#0 to both NUMA nodes using cudabw

6 C Programming API

API basics

- A hwloc program looks like this

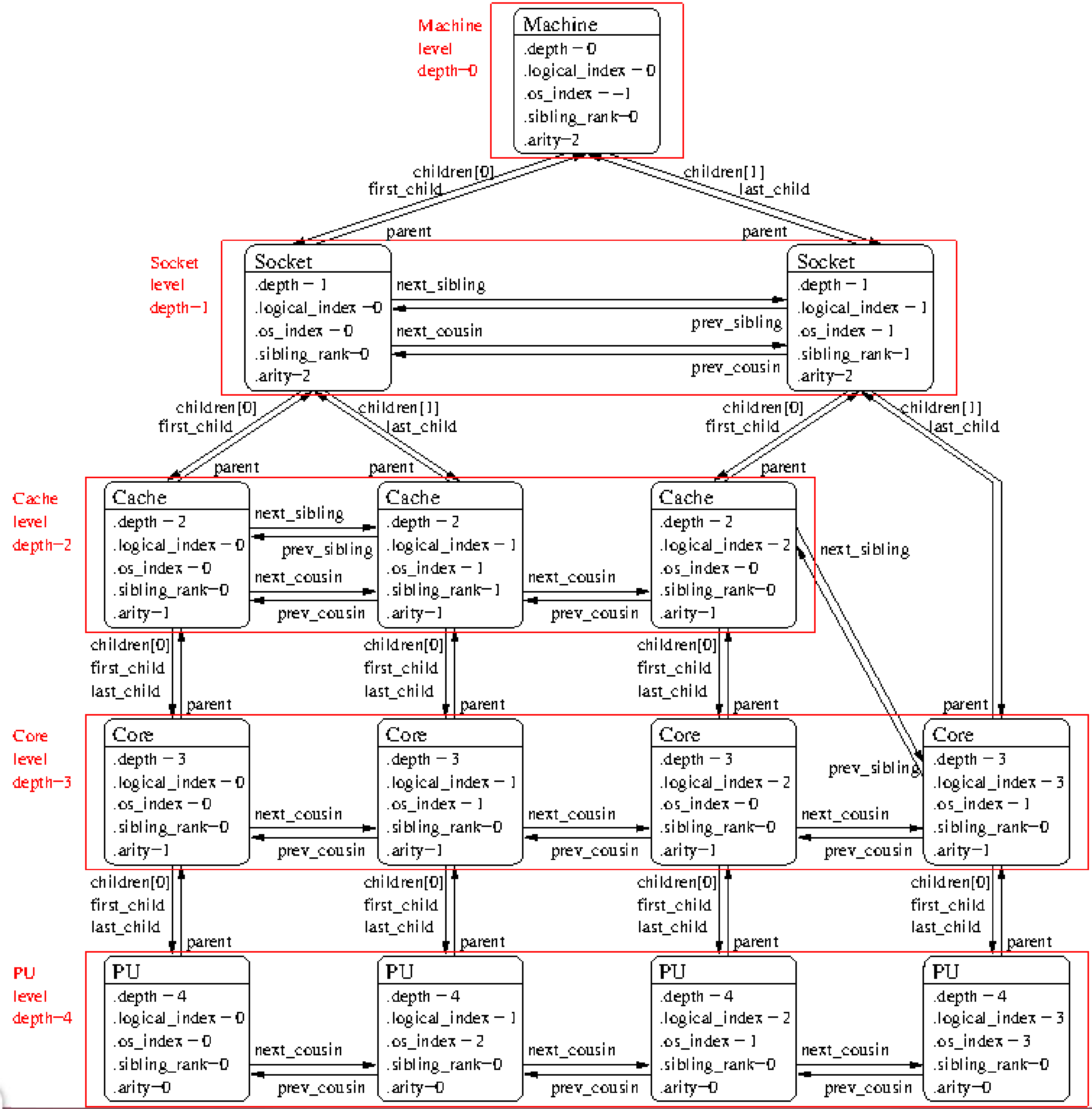
```
#include <hwloc.h>

hwloc_topology_t topo;

hwloc_topology_init(&topo);
/* ... configure what topology to build … */
hwloc_topology_load(topo);

/* … play with the topology … */

hwloc_topology_destroy(topo);
```

Major hwloc types

- The topology context : hwloc_topology_t
  - You always need one
- The main hwloc object : hwloc_obj_t
  - That's where the actual info is
  - The structure isn't opaque
    - It contains many pointers to ease traversal
- Object type : hwloc_obj_type_t
  - HWLOC_OBJ_PU, _PACKAGE, _CORE, _NUMANODE, …

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


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
Socket
.depth = 1
.logical_index = 1
.os_index = 1
.sibling_rank=1
.arity=2
next_sibling
prev_sibling
next_cousin
prev_cousin
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
children[0] first_child last_child
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

Browsing as a tree

- The root is hwloc_get_root_obj(topo)
- Objects have children
  - obj->arity is the number of children
  - The array of children is obj->children[]
  - They are also in a list
    - obj->first_child, obj->last_child
    - child->prev_sibling, child->next_sibling
    - NULL-terminated
- The parent is obj->parent (or NULL)

Browsing as levels

- The topology is also organized as levels of identical objects
  - Cores, L2d Caches, …
  - All PUs at the bottom
- Number of levels hwloc_topology_get_depth(topo)
- Number of objects on a level
  hwloc_get_nbobjs_by_type(topo, type)
  hwloc_get_nbobjs_by_depth(topo, depth)
- Convert between depth and type using
  hwloc_get_type_depth() or hwloc_get_depth_type()

Browsing as levels

- Find objects by level and index
  - hwloc_get_obj_by_type(topo, type, index)
  - There are variants taking a depth instead of a type
    - Note : the depth of my child is not always my depth + 1
      - Think of asymmetric topologies
- Iterate over objects of a level
  - Objects at the same levels are also interconnect by prev/next_cousin pointers
    - Don't mix up siblings (children list) and cousins (level)
  - hwloc_get_next_obj_by_type/depth()

Hands-on browsing the topology

Starting from basic.c

- Print the number of cores
- Print the type of the common ancestor of cores 0 and 2
- Print the memory size near core 0
- Iterate over all PUs and print their physical numbers

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

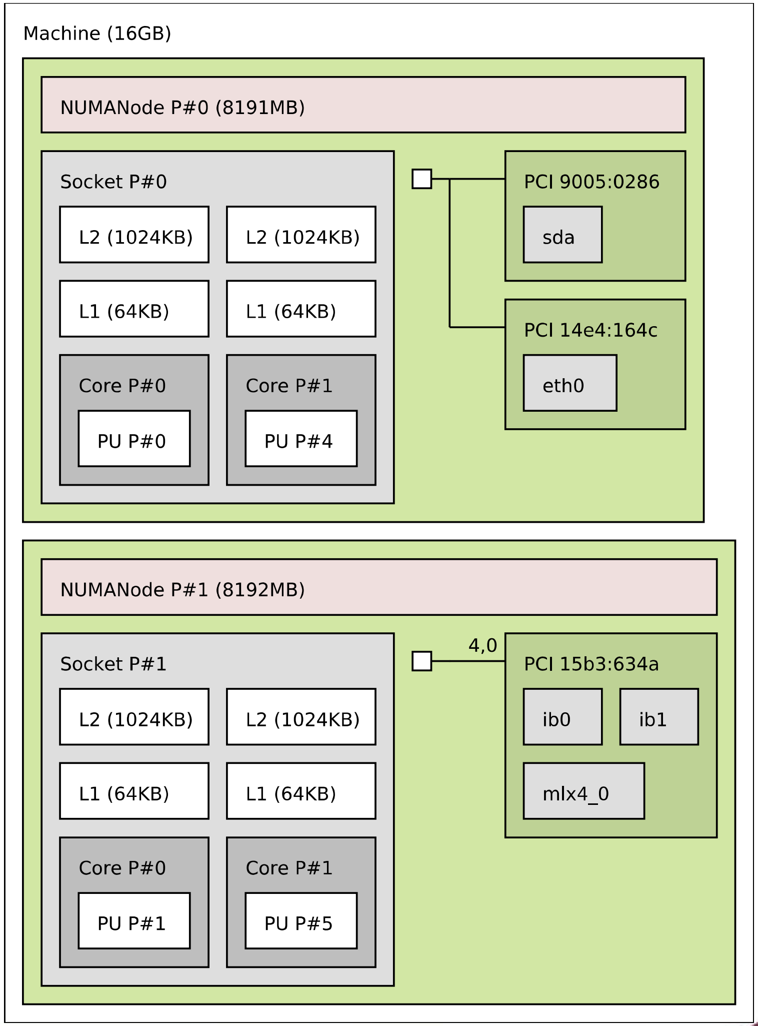
- Generic mask of bits : hwloc_bitmap_t
  - Possibly infinite
  - Opaque, used to describe object contents
    - Which PU are inside this object (obj->cpuset)
    - Which NUMA nodes are close to this object (obj->nodeset)
  - Can be combined to bind to multiple cores, etc.
    - and, or, xor, not, ...

Hands-on bitmaps and binding

- Bind a process to 1st core
- Rebind the same process to cores 2 and 4
- Print its binding
- Print where it's actually running
  - Repeat
- Rebind to avoid migrating between cores
  - hwloc_bitmap_singlify()

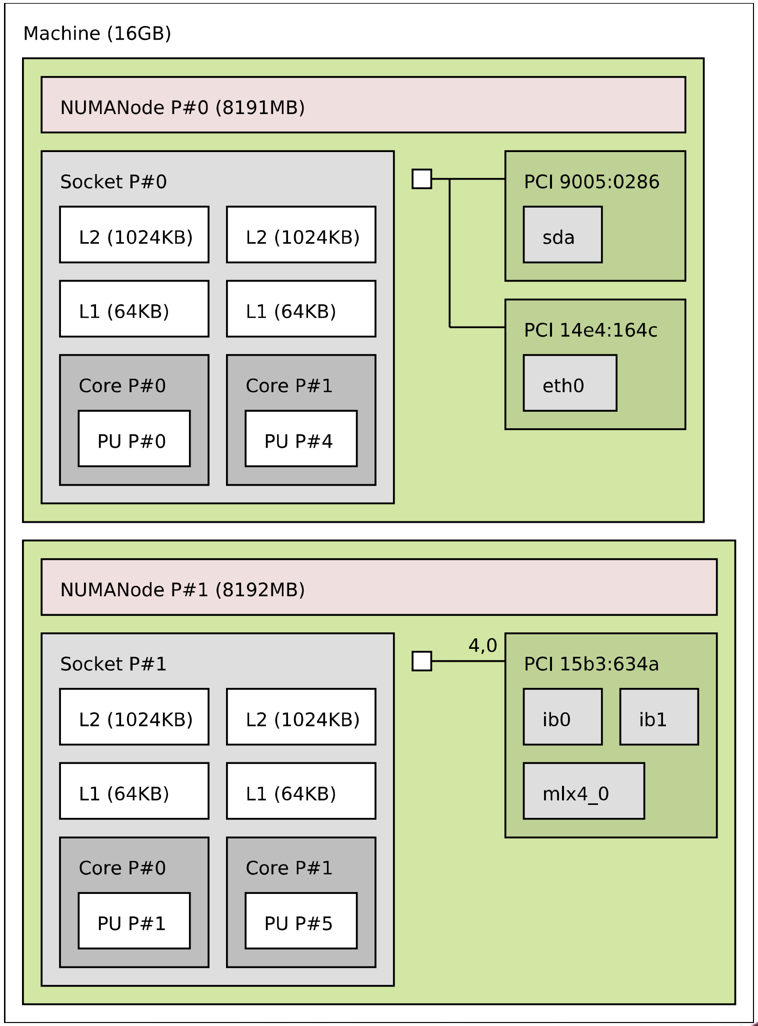
I/O devices

- 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



Hands-on I/O

- Load cuda modules, etc.

Starting from cuda.c

- Find the NUMA node near each CUDA device

Starting form ib.c

- Find the NUMA node near each IB device

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, …
    - See lstopo -v for (many) examples
  - You may add your own

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/helper.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

8 Conclusion

More information

- The documentation
  - http://www.open-mpi.org/projects/hwloc/doc/
  - Related pages
    - http://www.open-mpi.org/projects/hwloc/doc/v1.11.3/pages.php
  - FAQ
    - http://www.open-mpi.org/projects/hwloc/doc/v1.11.3/a00030.php
- 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://github.com/open-mpi/hwloc/issues for reporting bugs

Thanks!

Questions?

http://www.open-mpi.org/projects/hwloc

Brice.Goglin@inria.fr