HiPEAC
2013/01/21
Berlin

Understanding and managing hardware affinities with Hardware Locality (hwloc)

Brice Goglin

Resources for this tutorial

- During the tutorial
  - http://www.open-mpi.org/projects/hwloc
    - Google for hwloc
  - Click on the tutorial news on the right
- Later
  - From http://runtime.bordeaux.inria.fr/hwloc/tutorials
  - or Google for hwloc tutorials

Keep this webpage open for the entire day

- We'll download things from there

Agenda

- Introduction
- Hardware Locality presentation
- hwloc Installation
- Command-line Tools
- Programming API
- I/O Devices
- Miscellaneous features
- Conclusion

1 Introduction

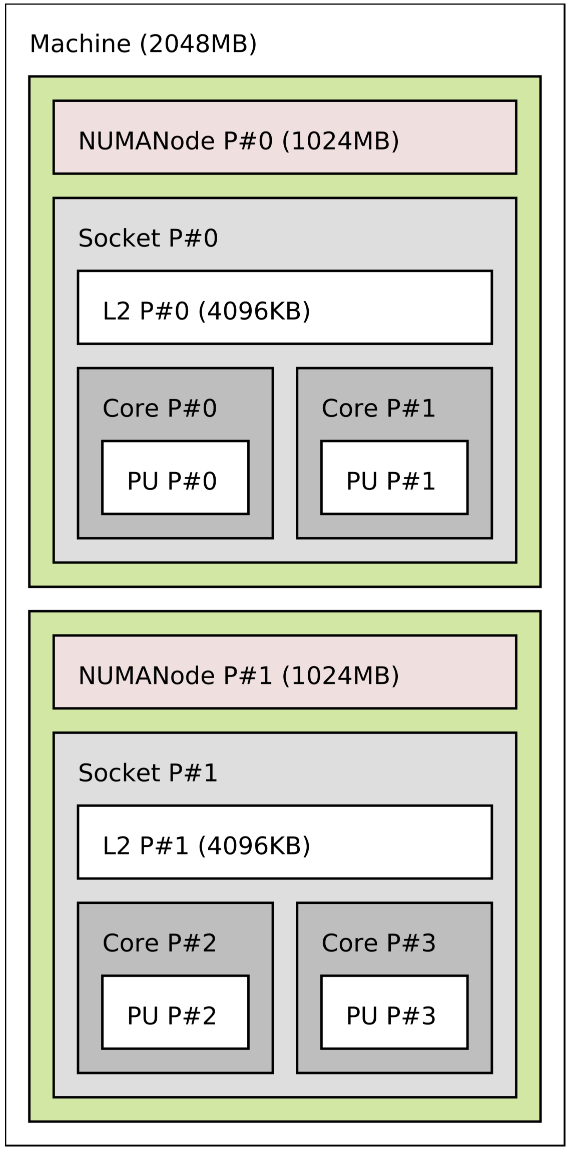
Machines are increasingly complex

Machines are increasingly complex

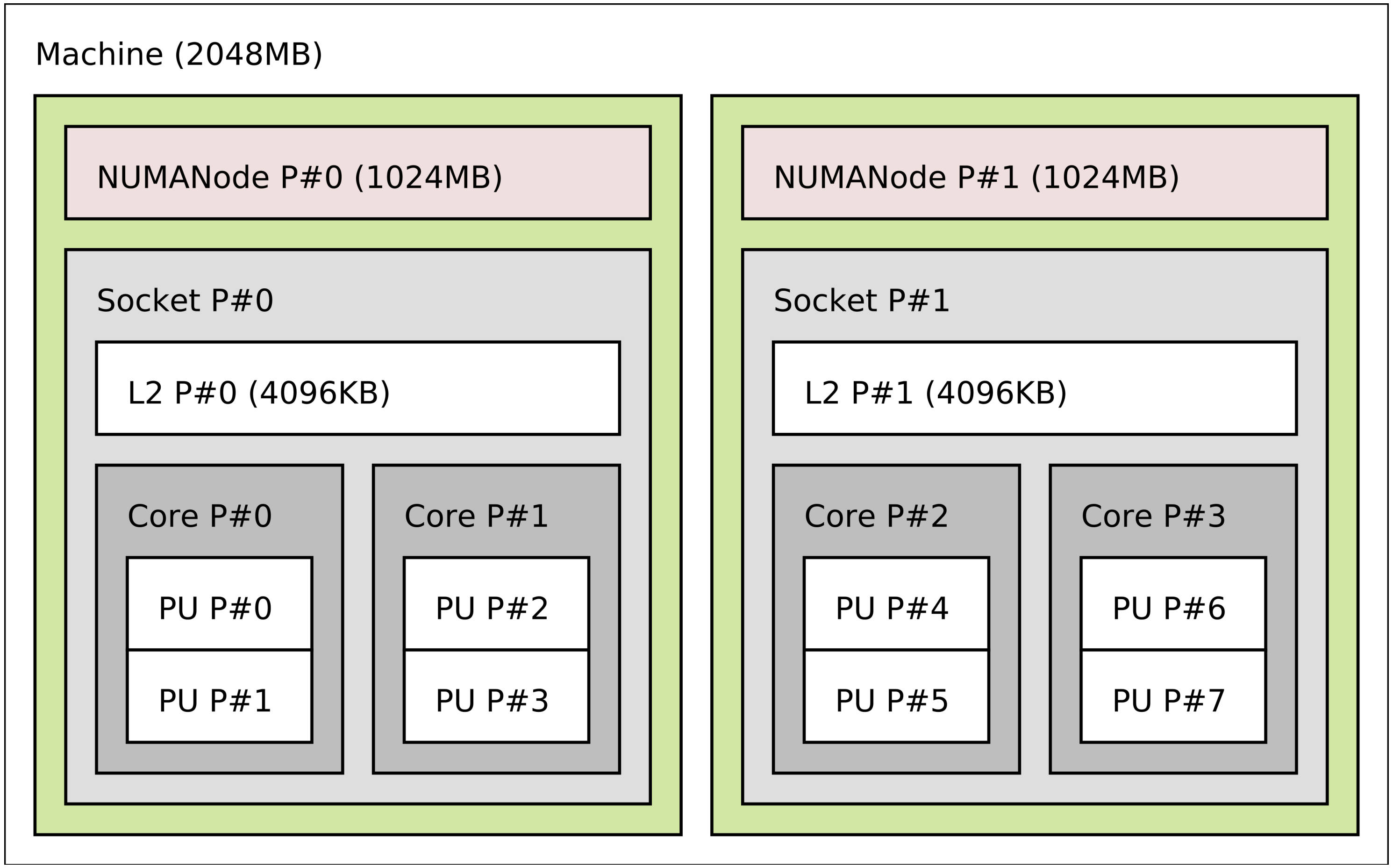
- Multiple processor sockets
- Multicore processors
- Simultaneous multithreading
- Shared caches
- NUMA
- GPUs, NICs, …
  - Close to some sockets (NUIOA)

Affinity are one of the key performance criteria

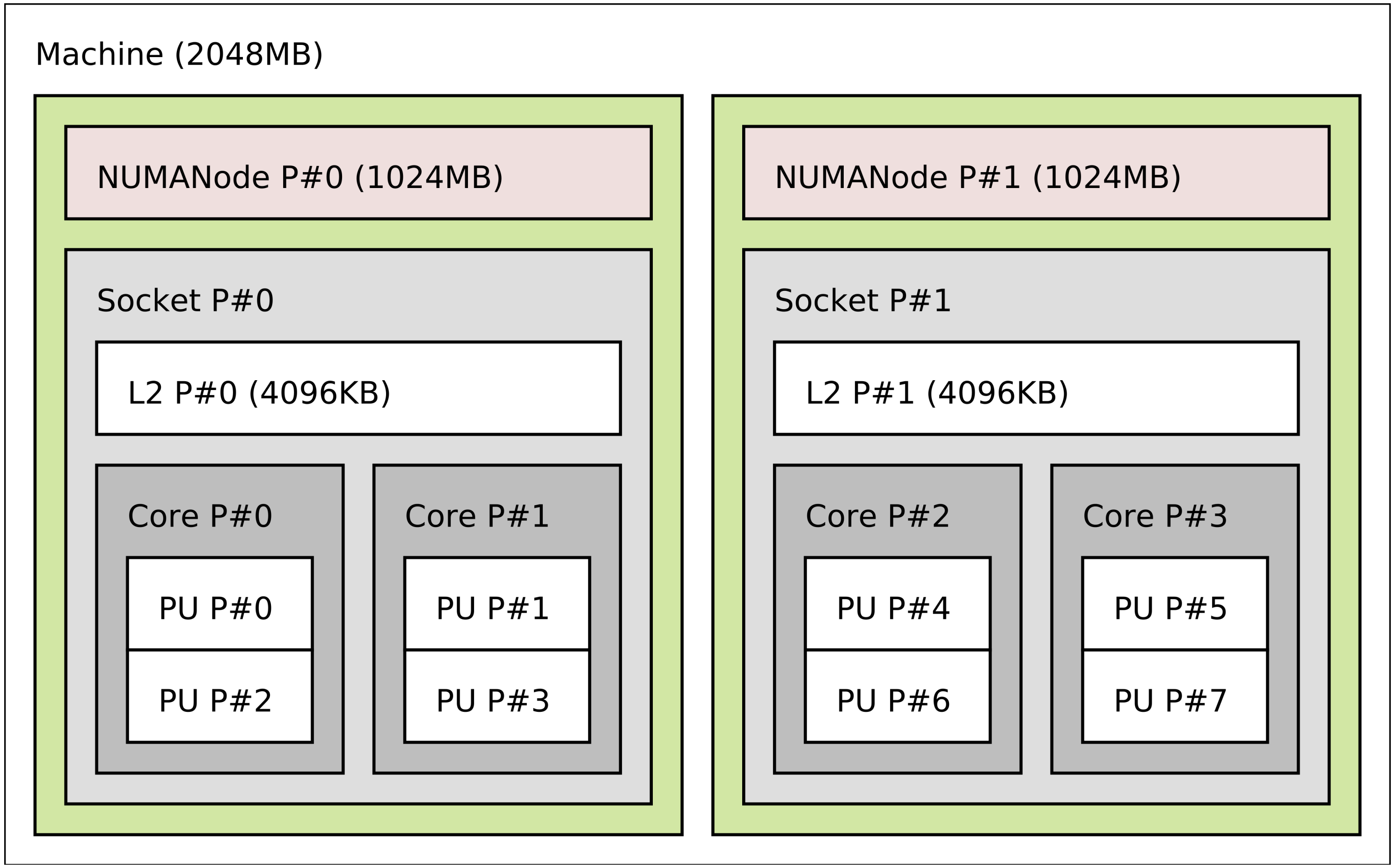
- Dilemma
  - Use cores 0 & 1 to share cache and improve synchronization cost ?
  - Use core 0 & 2 to maximize memory bandwidth ?
- Depends on
  - The machine structure
  - The application needs



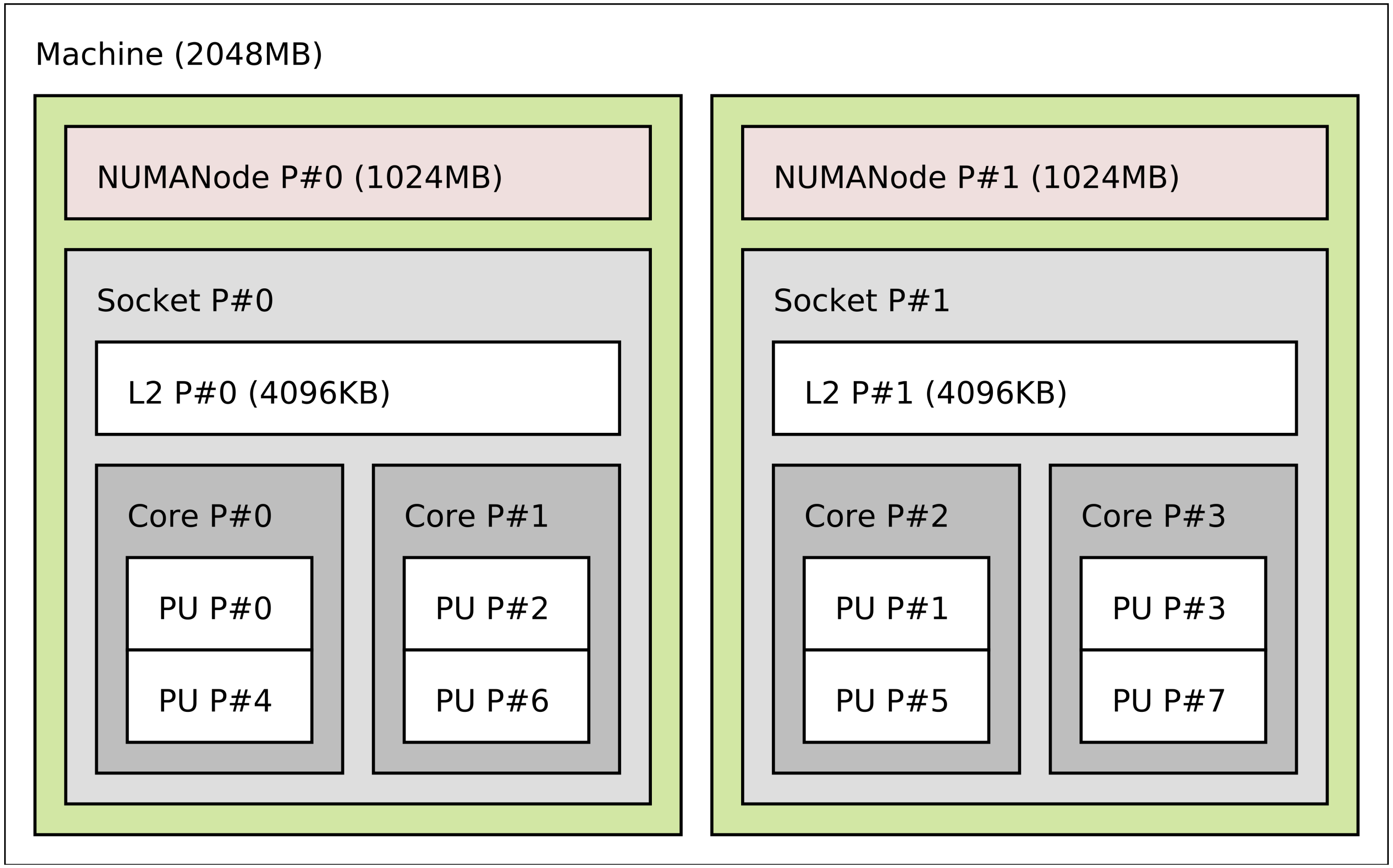
What's in my machine ?



Or maybe it's a bit different ?



Wait, after rebooting on another OS or BIOS ?



Hardware organization is unpredictable

- You may know what you bought
  - … but you can't assume how processors, cores, threads will be numbered
  - Depends on the vendor
  - Depends on the operating system
  - May change after BIOS update

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
  - Ordering of levels may change
    - Sockets may be inside NUMA nodes, or the contrary

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

2 Hardware Locality presentation

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

- 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 ICC
  - x86 specific, no API

(very quick) hwloc history

(the NEWS file contains much more than this)

- 2009/11 : hwloc v0.9.1 : first hwloc release, mostly only for topology detection
- 2010/05 : v1.0 : Process binding, XML
- 2010/12 : v1.1 : Memory binding, unlimited number of objects, annotable objects
- 2011/04 : v1.2 : distance API, get_last_cpu_location()
- 2011/10 : v1.3 : PCI and I/O objects
- 2012/01 : v1.4 : Multinode « custom » interface
- 2012/07 : v1.5 : Cache attributes
- 2012/12 : v1.6 : Plugins
- 2013/03 : v1.7 : CUDA, OpenCL, BlueGene/Q

Portability

- Linux
  - Supports almost everything
    - Not supported : Memory replication
- Solaris, AIX, HP-UX, OSF, *BSD, Windows
  - Topology detection sometimes limited
  - No I/O locality
- Darwin
  - No binding

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

Use case 1 : *TreeMatch* software
MPI process placement

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

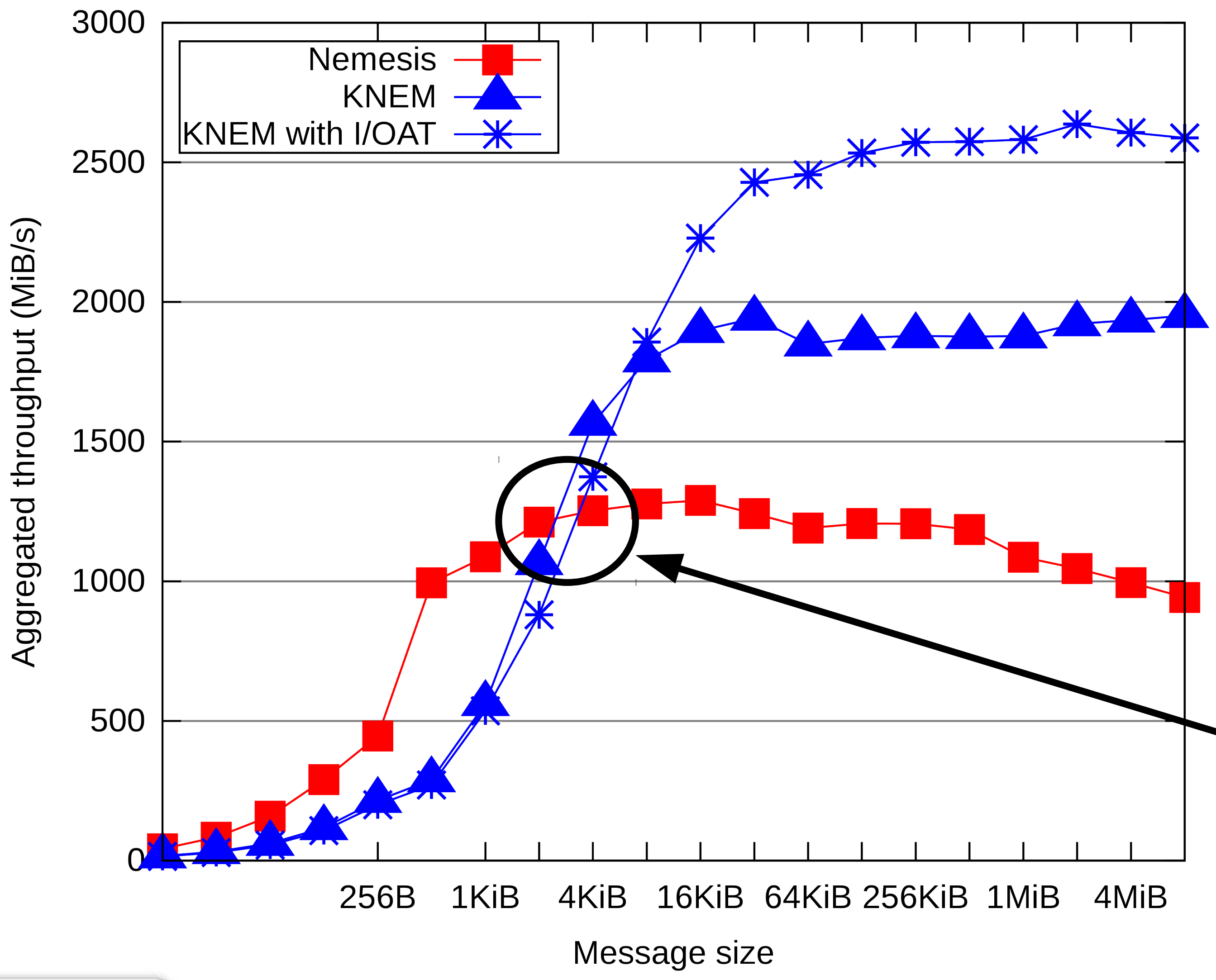
- This becomes a mapping of a tree of processes
  - Ordered by communication intensiveness
- … onto a tree of hardware resources
  - Given by hwloc

Use case 2 : *ForestGOMP* software
OpenMP thread scheduling

- 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

Use case 3 : Intra-node MPI data transfer

Topology-aware thresholds



Threshold that depends on shared cache size

3

hwloc Installation

Existing packages

- At least for Debian, Ubuntu, Redhat, Fedora, CentOS, ArchLinux, NetBSD
  - If recent enough (at least v1.3), just install it
- You want the development headers too
  - libhwloc-dev, hwloc-devel, ...

Requirements for manual installation

- On Linux, if the machine is NUMA, install the numactl/libnuma development headers
- If I/O devices matter, install the pciutils/libpci headers
- Add Cairo headers for lstopo graphics

- If building from SVN, see the README and HACKING files
  - Need recent autotools
  - You may want to use nightly tarballs instead

Manual installation

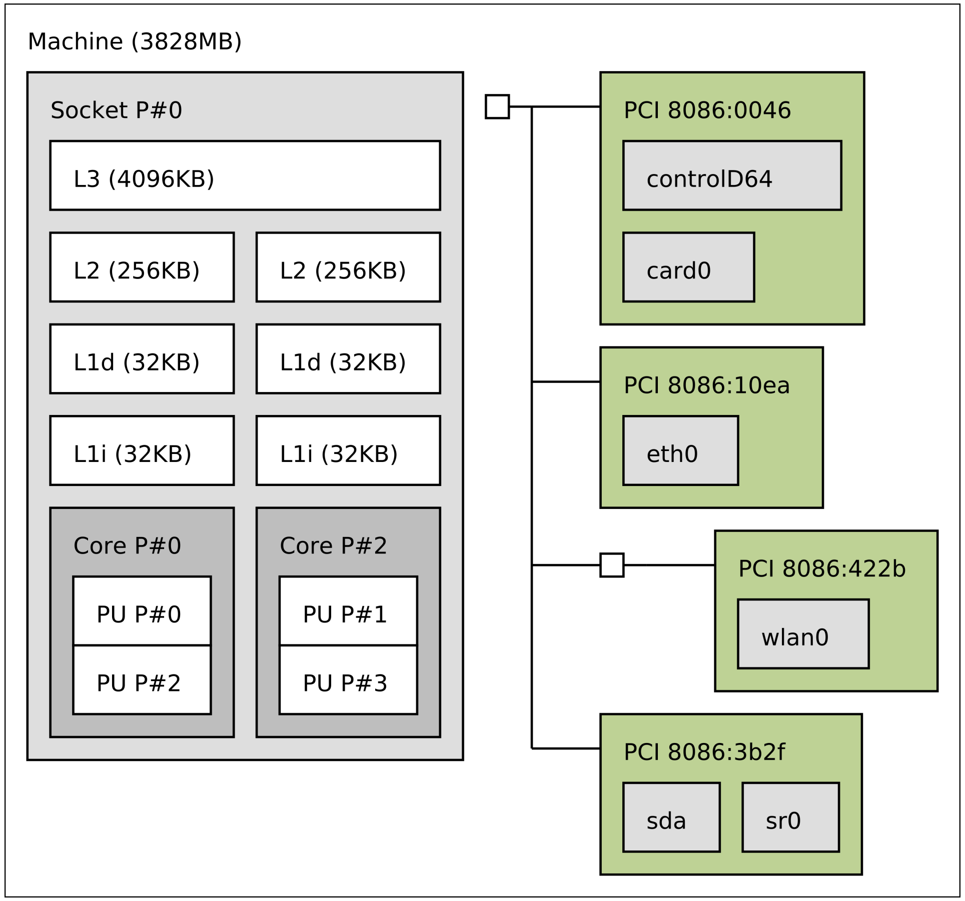
- Take a recent tarball at http://www.open-mpi.org/projects/hwloc
- ./configure --prefix=$PWD/install
  - Very few configure options
    - Disabling things (PCI, Cairo, …)
    - Enabling plugins (since v1.6, not needed here)
- Check the summary at the end of configure
  - PCI support isn't strictly required for this tutorial
    - But it would be nice to have

Manual installation

- make
  - Parallel builds supported, but the build is quick anyway
- make install
- Useful environment variables
  - export PATH=$PATH:<prefix>/bin
  - export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<prefix>/lib
  - export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:<prefix>/lib/pkgconfig
  - export MANPATH=$MANPATH:<prefix>/share/man
- Have access to a nice server for this tutorial ?
  - Install hwloc on the server AND on your local machine

4 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

- 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
      - Faster
  - Save for offline consultation
    - Batch schedulers placing processes on compute nodes
    - Remote debugging without access to the machine

Hands on lstopo

- Let's work locally first
- Basic graphic output

```
$ lstopo --no-io
```

- With I/O

```
$ lstopo
```

- Basic text output

```
$ lstopo --no-io -
```

- Verbose output (text by default, no merging)

```
$ lstopo -v
```

lstopo output formats

```
$ lstopo foo.png
$ lstopo foo.pdf
$ lstopo foo.fig (doesn't need Cairo)
```

- Export to stdout in a specific format

```
$ lstopo --of pdf
```

- Output format guessed from the extension

```
$ lstopo -.pdf > foo.pdf
```

lstopo and XML

- Dump a topology to a XML file

```
$ lstopo out.xml
```

- Reload it

```
$ lstopo --input out.xml --if xml
```

- Input format also guessed from the input name

```
$ lstopo -i out.xml
```

lstopo on a distant server

- Graphics across SSH may be slow
  - Put XML in the middle

```
remote$ lstopo foo.xml
local$ scp remote:foo.xml .
local$ lstopo -i foo.xml
```

- Or even easier

```
local$ ssh <remote> lstopo -.xml | lstopo --if xml -i -
```

lstopo for your slides and papers

- Need to draw your platform ?
  - lstopo has many configuration options
- --horiz and --vert to change the layout
- --ignore to remove useless levels
- --no-io, --no-icaches to ignore some objects
- --restrict to hide parts of the machines
- Synthetic topologies if you need a specific server

```
$ lstopo -i "node:4 socket:2 cache:1 core:4 pu:2"
```

- And a lot more, see lstopo --help

Hands on lstopo

- Create a topology containing
  - 2 NUMA nodes containing 2 sockets
  - 4 cores in each sockets, 2-way hyperthreaded
  - L3 shared by all cores, L2 by pairs, L1 not shared
- Save it to XML
- Reimport it and display it

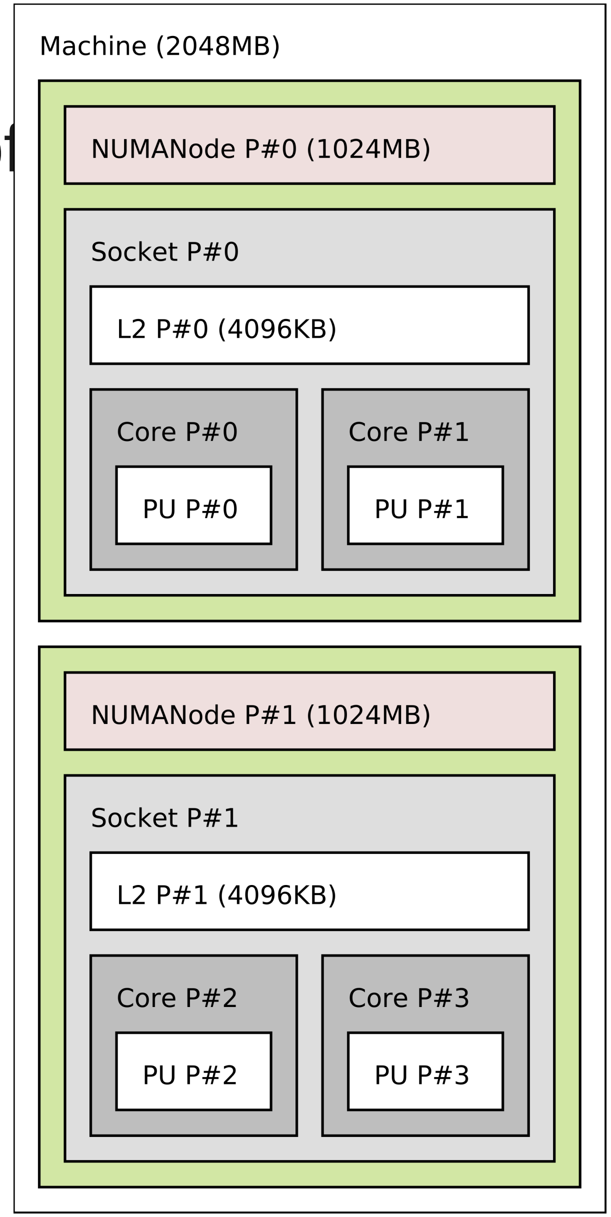
hwloc-calc – Compute CPU sets

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

```
$ hwloc-calc socket:1 ~pu:even
0x00000008
$ hwloc-calc socket:0.core:1
0x00000002
$ 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
```



Hands on hwloc-calc

- Reuse the previously saved XML topology
- Compute the bitmap containing the second socket
- Compute the bitmap containing
  - The third PU in second socket
  - and the first two cores in the second NUMA node
  - without the first PU on the second NUMA node
- Count the cores within second NUMA node, and list their IDs
- Display the topology restricted to the first socket

- On your machine
- Find the bitmap of CPUs near your network interface
- Display the list of PU, first by logical IDs, then by physical IDs (-p)

hwloc-bind – Bind processes and threads

- Bind a process to a given set of CPUs

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

- Bind an existing process

```
$ hwloc-bind --pid 1234 node:0
```

- Avoid migration within binding by adding --single
- Bind memory

```
$ hwloc-bind --membind node:1 --cpubind node:0 …
```

- Distribute memory

```
$ hwloc-bind --membind all --mempolicy interleave ...
```

Check binding

- hwloc-bind can tell where a process is bound and where it is actually running

```
$ hwloc-bind --pid 1234 --get
$ hwloc-bind --pid 1234 --get-last-cpu-location
```

- hwloc-ps can list bound processes and threads

```
$ hwloc-ps
$ hwloc-ps -t
```

- lstopo can display bound processes in the topology

```
$ lstopo --ps
```

Hands on hwloc-bind

- Generate the list of core IDs of your local machine using hwloc-calc --sep " "
- Use the output in a loop to launch/bind a "sleep 1000" on each core
- Display these process binding with lstopo --ps and hwloc-ps
- Move one process to another core and display again
- Rebind one process to the entire machine and display again
- Use --get-last-cpu-location to see where it actually runs

Other tools

- Get some object information
  - hwloc-info (starting in hwloc v1.7)
- Assemble multiple topologies from different nodes
  - hwloc-assembler and hwloc-assembler-remote
- Display distance matrices
  - hwloc-distances
- Generate bitmaps for distributing multiple processes on a topology
  - hwloc-distrib
- Save a Linux node topology info for debugging
  - hwloc-gather-topology

5

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);
```

Building programs using hwloc

- Download Makefile and open it
- pkg-config may be used to find headers and libraries
  - But CFLAGS and LDFLAGS are also easy to set manually
- Download basic.c and compile it
  - We'll use this program as the base for later examples
- Now display the number of cores using

```
hwloc_get_nbobjs_by_type(topo, HWLOC_OBJ_CORE)
```

Major hwloc types

- The topology context : hwloc_topology_t
  - You always need one
    - Except when only playing with bitmaps (see later)
- 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, _CORE, _NODE, …

Browsing

- hwloc objects are interconnected in many ways to ease browsing
- All links are described in
  - http://www.open-mpi.org/projects/hwloc/doc/v1.6/diagram.png
- Many terms are explained in
  - http://www.open-mpi.org/projects/hwloc/doc/v1.6/a00001.php

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)

Hands on tree browsing

- Write a function that takes an object and prints its type, depth and os_index
- Call it on the root object of the topology
- Modify the function to later call itself on each children
  - Once with the obj->children[] array
  - Once with the list of children/siblings
- Write a function that checks whether obj2 is an ancestor of obj1 by walking up the parent links
  - Test it on the first PU and the root object

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 level browsing

- Display the first object of each level
- Display all objects of the PU level
  - Using get_obj_by_type()
- Display all objects of the last level
  - Using a loop of get_next_obj_by_depth()

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/moon
- 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
- Physical index bits are hidden in bitmap bits
  - You don't care what they actually mean, you just use obj->cpuset and so on
  - That's what hwloc-calc displays by default

Bitmap, CPU sets, Node sets

- Generic mask of bits : hwloc_bitmap_t
  - Possibly infinite
  - Used to describe object contents
    - Set of bits identifying the PUs included in an object
      - hwloc_cpuset_t is a synonym
    - Set of bits identifying the NUMA nodes near an object
      - hwloc_nodeset_t is a synonym
    - May be used to store whatever you need

Manipulating bitmaps

- Don't ever modify obj->cpuset or obj->nodeset
- Duplicate one with hwloc_bitmap_dup()
  or create a new one with hwloc_bitmap_alloc()
  - And destroy it with hwloc_bitmap_free()
- hwloc/bitmap.h offers many operations
  - And/Or/Xor/Not
  - Fill/Zero
  - Comparison
  - Finding first/last/next/number-of bits
  - Singlification (useful before binding)
  - Stringification (useful for debugging)

Hands on bitmaps

- Create a bitmap containing the cpuset of the first and last PU objects
- Display it

- Read a line from stdin and convert it into a bitmap
- Iterate over cores and display all the ones that intersect the bitmap

CPU Binding API

- Bind the current process or thread
  - hwloc_set_cpubind(topo, cpuset, flags)
    - flags is HWLOC_CPUBIND_THREAD or PROCESS
      - 0 if single-thread process
    - More flags for more precise behavior
  - hwloc_get_cpubind() for retrieving current binding
- For another process or thread
  - hwloc_set/get_proc/thread_cpubind()
- The cpuset argument is usually built from obj->cpuset

Hands on CPU binding

- Bind the current process on the last core
- Create a pthread that sleeps for 1 second
- Have the master thread bind it to the first core
- Then the thread prints its own binding and current CPU location, and the entire process binding and current CPU location
- Then the thread rebinds itself on a single PU of the last core, and prints all this again
- Before the end, the main thread prints all this again

- If your machine isn't hyperthreaded, find one with two sockets and replace core with socket in all the text above

Memory Binding API

- Allocating memory on specific memory nodes
  - hwloc_alloc_membind_policy()
- Changing the allocation policy of a process
  - Or of an existing memory zone
- Many memory placement policies/flags
  - First touch, next touch, force bind, interleave, replicate
    - When supported by the OS
      - hwloc_topology_get_support() tells you what is supported
  - Migrate if already allocated on a different node
- The nodeset argument is usually built from obj->nodeset

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

Interoperability helpers

- When you use other libraries
  - Different structures for sets of CPUs ?
    - glibc sched.h CPU sets, numactl nodemasks, …
    - Helpers to convert between these and hwloc bitmaps
  - Misc software handles
    - OpenFabrics Verbs devices, CUDA devices, …
    - Helpers to retrieve their locality
- And some Linux specific helpers
  - Binding threads by TID
- http://www.open-mpi.org/projects/hwloc/doc/v1.6/a00010.php

XML API

- Exporting a topology to a XML file
  - hwloc_topology_export_xml(topo, filename)
- Importing from a XML file
  - hwloc_topology_set_xml(topo, filename)
    - To be placed between init() and load()
- « xmlbuffer » variants
  - Useful for passing topologies between processes
    - On the network, ...

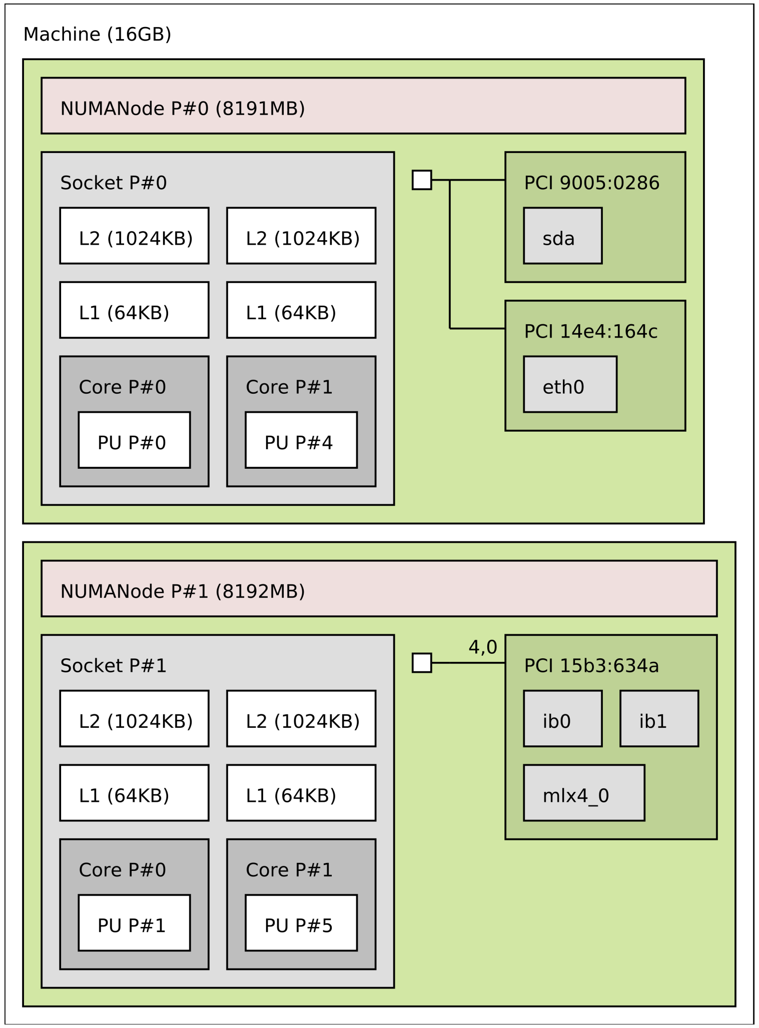
6 I/O Devices

Why and how

- 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
  - No corresponding bits in the cpuset and nodeset
    - But a cpuset defining which CPUs and nodes are close
  - 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
    - Topology flags
- 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



Current status

- PCI discovery with pciutils/libpci
  - Gives PCI bridges and buses
  - Available on most Unixes
    - Not on Darwin and Windows
  - May require admin privileges
    - Ask your admin to export to XML !
- PCI locality only available on Linux
- OS devices discovery
  - Disks, NICs, InfiniBand, ... on Linux
  - AMD OpenCL, NVIDIA CUDA/NVML GPUs in v1.7

Consulting I/O object

- Special levels, depths and iterators
  - HWLOC_OBJ_PCI_DEVICE
  - HWLOC_TYPE_DEPTH_PCI_DEVICE
  - hwloc_get_next_pcidev(topo, prevobj)
  - Same things for OS devices (and bridges)
- The locality is in parents
  - Walk up the obj->parent pointer
    until obj->cpuset isn't NULL
    - hwloc_get_non_io_ancestor_obj(topo, ioobj)

Hands on I/O objects

- List PCI objects, print their PCI bus ID, name and locality

- List OS devices, print their type, name and locality

I/O affinity without objects

- Sometimes you don't want I/O objects
  - If you just need their locality, no attributes
  - If they are not well supported
- hwloc interoperability helpers can help
  - hwloc/cuda.h and hwloc/cudart.h return the locality (cpuset) of NVIDIA devices
  - hwloc/openfabrics-verbs.h return the locality of IB HCAs
  - Many more, see http://www.open-mpi.org/projects/hwloc/doc/v1.6/a00010.php

7

Miscellaneous features

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
    - Need to export/import it to XML ? Define some callbacks
- (name,value) info attributes
  - Basic string annotations, hwloc adds some
    - Backend name, CPU Model, PCI Vendor, ...
  - You may add more
  - Already exported/imported to XML

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...

Distances

- hwloc gathers NUMA distances from the BIOS
  - And the user may add some custom matrices
- Used internally for grouping objects by distance
  - e.g. 4 groups of 4 nodes instead of 16 nodes
- The application may consult them
  - Object distance attributes

« Custom » API and Tools

- A topology may contain a System root object with multiple Machine children
  - Multi-node topology
- The « Custom » API lets you assembles multiple topologies into a single one
  - Insert objects and topologies into an empty one before load()
  - Be careful when binding!
- hwloc-assembler and hwloc-assembler-remote command-line tools

Binding and XML

- When you load a XML topology, hwloc doesn't know if it matches the local node
  - Binding is disabled by default
    - The number and types of CPUs may be different
- May be reverting by setting a topology flag
  - HWLOC_TOPOLOGY_FLAG_IS_THISSYSTEM
    - « Don't worry, I guarantee this is the local machine »

8

Conclusion

More information

- The documentation
  - http://www.open-mpi.org/projects/hwloc/doc/
  - Related pages
    - http://www.open-mpi.org/projects/hwloc/doc/v1.6/pages.php
  - FAQ
    - http://www.open-mpi.org/projects/hwloc/doc/v1.6/a00014.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://svn.open-mpi.org/trac/hwloc/report for reporting bugs

Thanks !

Brice.Goglin@inria.fr