# Hardware Locality (hwloc)

v2.2-20200403.0300.gitd9afcd5

Generated by Doxygen 1.6.1

Fri Apr 3 03:01:22 2020

# Contents

# Chapter 1

# Hardware Locality

## Portable abstraction of parallel architectures for high-performance computing

## 1.1 Introduction

The Hardware Locality (hwloc) software project aims at easing the process of discovering hardware resources in parallel architectures. It offers command-line tools and a C API for consulting these resources, their locality, attributes, and interconnection. hwloc primarily aims at helping high-performance computing (HPC) applications, but is also applicable to any project seeking to exploit code and/or data locality on modern computing platforms.

hwloc is actually made of two subprojects distributed together:

- **The original hwloc project for describing the internals of computing nodes**. It is described in details starting at section Hardware Locality (hwloc) Introduction.
- **The network-oriented companion called netloc (Network Locality)**, described in details starting with section Network Locality (netloc).

Netloc may be disabled, but the original hwloc cannot. Both hwloc and netloc APIs are documented after these sections.

## 1.2 Installation

hwloc (https://www.open-mpi.org/projects/hwloc/) is available under the BSD license. It is hosted as a sub-project of the overall Open MPI project (https://www.open-mpi.org/). Note that hwloc does not require any functionality from Open MPI -- it is a wholly separate (and much smaller!) project and code base. It just happens to be hosted as part of the overall Open MPI project.

### 1.2.1 Basic Installation

Installation is the fairly common GNU-based process:

```
shell$ ./configure --prefix=...
```

```
shell$ make
shell$ make install
```

hwloc- and netloc-specific configure options and requirements are documented in sections hwloc Installation and Netloc Installation respectively.

Also note that if you install supplemental libraries in non-standard locations, hwloc's configure script may not be able to find them without some help. You may need to specify additional CPPFLAGS, LDFLAGS, or PKG_CONFIG_PATH values on the configure command line.

For example, if libpciaccess was installed into /opt/pciaccess, hwloc's configure script may not find it be default. Try adding PKG_CONFIG_PATH to the ./configure command line, like this:

```
./configure PKG_CONFIG_PATH=/opt/pciaccess/lib/pkgconfig ...
```

Running the "lstopo" tool is a good way to check as a graphical output whether hwloc properly detected the architecture of your node. Netloc command-line tools can be used to display the network topology interconnecting your nodes.

### 1.2.2 Installing from a Git clone

Additionally, the code can be directly cloned from Git:

```
shell$ git clone https://github.com/open-mpi/hwloc.git
shell$ cd hwloc
shell$ ./autogen.sh
```

Note that GNU Autoconf >=2.63, Automake >=1.11 and Libtool >=2.2.6 are required when building from a Git clone.

Nightly development snapshots are available on the web site, they can be configured and built without any need for Git or GNU Autotools.

## 1.3 Questions and Bugs

Bugs should be reported in the tracker (https://github.com/open-mpi/hwloc/issues). Opening a new issue automatically displays lots of hints about how to debug and report issues.

Questions may be sent to the users or developers mailing lists (https://www.open-mpi.org/community/lists/hwloc.php).

There is also a #hwloc IRC channel on Freenode (irc.freenode.net).

# Chapter 2

# Hardware Locality (hwloc) Introduction

# Portable abstraction of hierarchical architectures for high-performance computing

See also Further Reading for links to more sections about hwloc concepts.

## 2.1 hwloc Summary

hwloc provides command line tools and a C API to obtain the hierarchical map of key computing elements within a node, such as: NUMA memory nodes, shared caches, processor packages, dies and cores, processing units (logical processors or "threads") and even I/O devices. hwloc also gathers various attributes such as cache and memory information, and is portable across a variety of different operating systems and platforms.

hwloc primarily aims at helping high-performance computing (HPC) applications, but is also applicable to any project seeking to exploit code and/or data locality on modern computing platforms.

hwloc supports the following operating systems:

- Linux (including old kernels not having sysfs topology information, with knowledge of cpusets, ScaleMP vSMP support, etc.) on all supported hardware, including Intel Xeon Phi and NumaScale NumaConnect.
- Solaris (with support for processor sets and logical domains)
- AIX
- Darwin / OS X
- FreeBSD and its variants (such as kFreeBSD/GNU)
- NetBSD
- HP-UX
- Microsoft Windows
- IBM BlueGene/Q Compute Node Kernel (CNK)

Since it uses standard Operating System information, hwloc's support is mostly independant from the processor type (x86, powerpc, ...) and just relies on the Operating System support. The main exception is BSD operating systems (NetBSD, FreeBSD, etc.) because they do not provide support topology information, hence hwloc uses an x86-only CPUID-based backend (which can be used for other OSes too, see the Components and plugins section).

To check whether hwloc works on a particular machine, just try to build it and run `lstopo` or `lstopo-no-graphics`. If some things do not look right (e.g. bogus or missing cache information), see Questions and Bugs.

hwloc only reports the number of processors on unsupported operating systems; no topology information is available.

For development and debugging purposes, hwloc also offers the ability to work on "fake" topologies:

- Symmetrical tree of resources generated from a list of level arities, see Synthetic topologies.
- Remote machine simulation through the gathering of topology as XML files, see Importing and exporting topologies from/to XML files.

hwloc can display the topology in a human-readable format, either in graphical mode (X11), or by exporting in one of several different formats, including: plain text, PDF, PNG, and FIG (see Command-line Examples below). Note that some of the export formats require additional support libraries.

hwloc offers a programming interface for manipulating topologies and objects. It also brings a powerful CPU bitmap API that is used to describe topology objects location on physical/logical processors. See the Programming Interface below. It may also be used to binding applications onto certain cores or memory nodes. Several utility programs are also provided to ease command-line manipulation of topology objects, binding of processes, and so on.

Perl bindings are available from Bernd Kallies on CPAN.

Python bindings are available from Guy Streeter:

- Fedora RPM and tarball.
- git tree (html).

## 2.2 hwloc Installation

The generic installation procedure for both hwloc and netloc is described in Installation.

The hwloc command-line tool "lstopo" produces human-readable topology maps, as mentioned above. It can also export maps to the "fig" file format. Support for PDF, Postscript, and PNG exporting is provided if the "Cairo" development package (usually `cairo-devel` or `libcairo2-dev`) can be found in "lstopo" when hwloc is configured and build.

The hwloc core may also benefit from the following development packages:

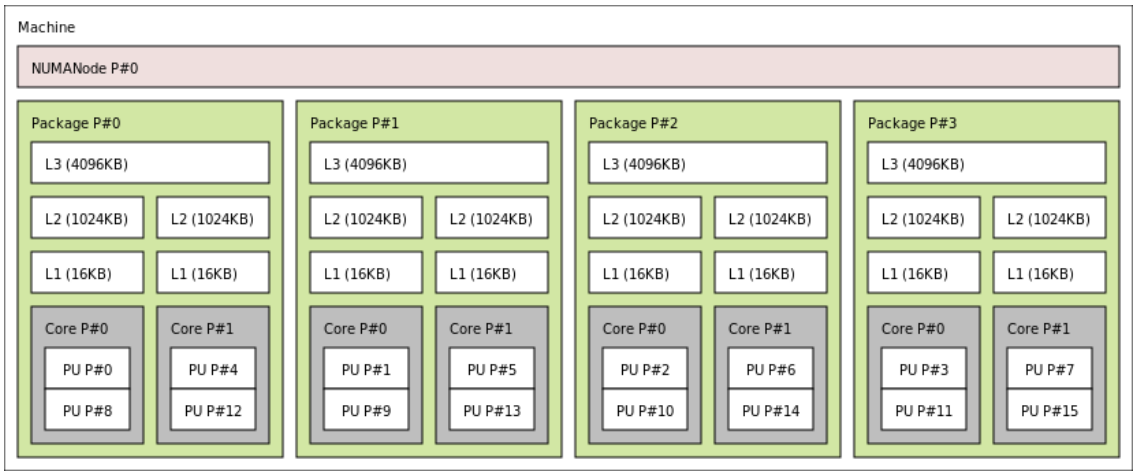
- libpciaccess for full I/O device discovery (`libpciaccess-devel` or `libpciaccess-dev` package). On Linux, PCI discovery may still be performed (without vendor/device names) even if libpciaccess cannot be used.
- AMD or NVIDIA OpenCL implementations for OpenCL device discovery.
- the NVIDIA CUDA Toolkit for CUDA device discovery.
- the NVIDIA Management Library (NVML) for NVML device discovery. It is included in CUDA since version 8.0. Older NVML releases were available within the NVIDIA GPU Deployment Kit from https://developer.nvidia.com/gpu-deployment-kit .
- the NV-CONTROL X extension library (NVCtrl) for NVIDIA display discovery. The relevant development package is usually `libXNVCtrl-devel` or `libxnvctrl-dev`. It is also available within nvidia-settings from ftp://download.nvidia.com/XFree86/nvidia-settings/ and https://github.com/NVIDIA/nvidia-settings/ .
- libxml2 for full XML import/export support (otherwise, the internal minimalistic parser will only be able to import XML files that were exported by the same hwloc release). See Importing and exporting topologies from/to XML files for details. The relevant development package is usually `libxml2-devel` or `libxml2-dev`.
- libudev on Linux for easier discovery of OS device information (otherwise hwloc will try to manually parse udev raw files). The relevant development package is usually `libudev-devel` or `libudev-dev`.
- libtool's ltdl library for dynamic plugin loading if the native dlopen cannot be used. The relevant development package is usually `libtool-ltdl-devel` or `libltdl-dev`.

PCI and XML support may be statically built inside the main hwloc library, or as separate dynamically-loaded plugins (see the Components and plugins section).

Note that because of the possibility of GPL taint, the `pciutils` library `libpci` will not be used (remember that hwloc is BSD-licensed).

## 2.3 Command-line Examples

On a 4-package 2-core machine with hyper-threading, the `lstopo` tool may show the following graphical output:

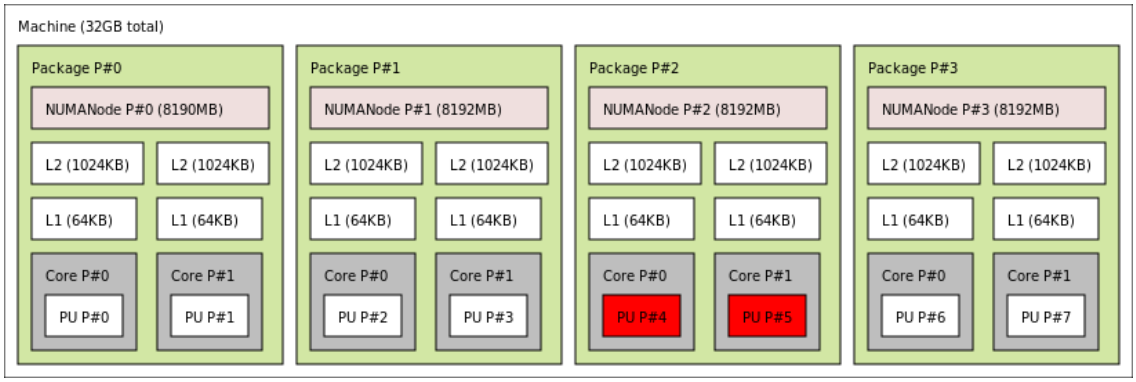



Here's the equivalent output in textual form:

```
Machine
  NUMANode L#0 (P#0)
  Package L#0 + L3 L#0 (4096KB)
    L2 L#0 (1024KB) + L1 L#0 (16KB) + Core L#0
      PU L#0 (P#0)
      PU L#1 (P#8)
    L2 L#1 (1024KB) + L1 L#1 (16KB) + Core L#1
      PU L#2 (P#4)
      PU L#3 (P#12)
  Package L#1 + L3 L#1 (4096KB)
    L2 L#2 (1024KB) + L1 L#2 (16KB) + Core L#2
      PU L#4 (P#1)
      PU L#5 (P#9)
    L2 L#3 (1024KB) + L1 L#3 (16KB) + Core L#3
      PU L#6 (P#5)
      PU L#7 (P#13)
  Package L#2 + L3 L#2 (4096KB)
    L2 L#4 (1024KB) + L1 L#4 (16KB) + Core L#4
      PU L#8 (P#2)
      PU L#9 (P#10)
    L2 L#5 (1024KB) + L1 L#5 (16KB) + Core L#5
      PU L#10 (P#6)
      PU L#11 (P#14)
  Package L#3 + L3 L#3 (4096KB)
    L2 L#6 (1024KB) + L1 L#6 (16KB) + Core L#6
      PU L#12 (P#3)
      PU L#13 (P#11)
    L2 L#7 (1024KB) + L1 L#7 (16KB) + Core L#7
      PU L#14 (P#7)
      PU L#15 (P#15)
```

Note that there is also an equivalent output in XML that is meant for exporting/importing topologies but it is hardly readable to human-beings (see Importing and exporting topologies from/to XML files for details).

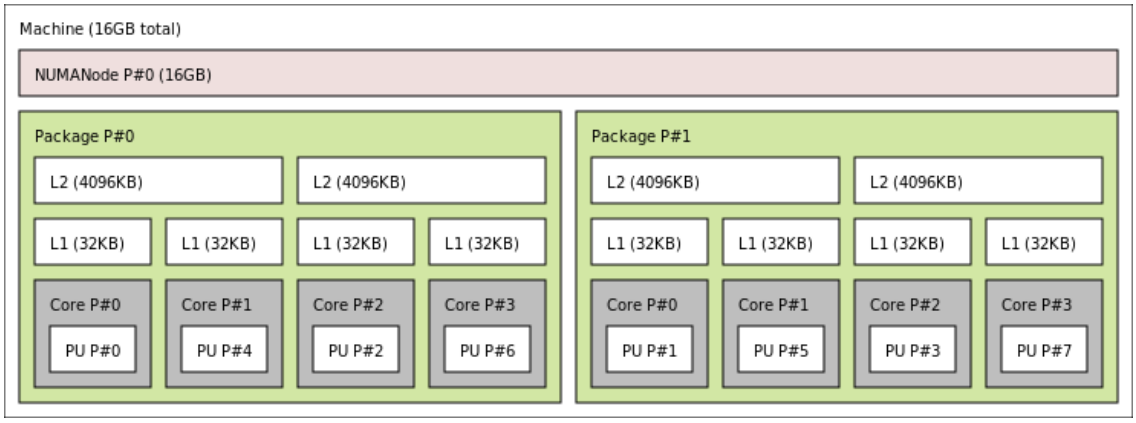
On a 4-package 2-core Opteron NUMA machine (with two core cores disallowed by the administrator), the `lstopo` tool may show the following graphical output (with `\--disallowed` for displaying disallowed objects):




Here's the equivalent output in textual form:

```
Machine (32GB total)
  Package L#0
    NUMANode L#0 (P#0 8190MB)
    L2 L#0 (1024KB) + L1 L#0 (64KB) + Core L#0 + PU L#0 (P#0)
    L2 L#1 (1024KB) + L1 L#1 (64KB) + Core L#1 + PU L#1 (P#1)
  Package L#1
    NUMANode L#1 (P#1 8192MB)
    L2 L#2 (1024KB) + L1 L#2 (64KB) + Core L#2 + PU L#2 (P#2)
    L2 L#3 (1024KB) + L1 L#3 (64KB) + Core L#3 + PU L#3 (P#3)
  Package L#2
    NUMANode L#2 (P#2 8192MB)
    L2 L#4 (1024KB) + L1 L#4 (64KB) + Core L#4 + PU L#4 (P#4)
    L2 L#5 (1024KB) + L1 L#5 (64KB) + Core L#5 + PU L#5 (P#5)
  Package L#3
    NUMANode L#3 (P#3 8192MB)
    L2 L#6 (1024KB) + L1 L#6 (64KB) + Core L#6 + PU L#6 (P#6)
    L2 L#7 (1024KB) + L1 L#7 (64KB) + Core L#7 + PU L#7 (P#7)
```

On a 2-package quad-core Xeon (pre-Nehalem, with 2 dual-core dies into each package):




Here's the same output in textual form:

```
Machine (total 16GB)
  NUMANode L#0 (P#0 16GB)
  Package L#0
    L2 L#0 (4096KB)
      L1 L#0 (32KB) + Core L#0 + PU L#0 (P#0)
      L1 L#1 (32KB) + Core L#1 + PU L#1 (P#4)
    L2 L#1 (4096KB)
      L1 L#2 (32KB) + Core L#2 + PU L#2 (P#2)
      L1 L#3 (32KB) + Core L#3 + PU L#3 (P#6)
  Package L#1
    L2 L#2 (4096KB)
      L1 L#4 (32KB) + Core L#4 + PU L#4 (P#1)
      L1 L#5 (32KB) + Core L#5 + PU L#5 (P#5)
    L2 L#3 (4096KB)
      L1 L#6 (32KB) + Core L#6 + PU L#6 (P#3)
      L1 L#7 (32KB) + Core L#7 + PU L#7 (P#7)
```

## 2.4 Programming Interface

The basic interface is available in hwloc.h. Some higher-level functions are available in hwloc/helper.h to reduce the need to manually manipulate objects and follow links between them. Documentation for all these is provided later in this document. Developers may also want to look at hwloc/inlines.h which contains the actual inline code of some hwloc.h routines, and at this document, which provides good higher-level topology traversal examples.

To precisely define the vocabulary used by hwloc, a Terms and Definitions section is available and should probably be read first.

Each hwloc object contains a cpuset describing the list of processing units that it contains. These bitmaps may be used for CPU binding and Memory binding. hwloc offers an extensive bitmap manipulation interface in hwloc/bitmap.h.

Moreover, hwloc also comes with additional helpers for interoperability with several commonly used environments. See the Interoperability With Other Software section for details.

The complete API documentation is available in a full set of HTML pages, man pages, and self-contained PDF files (formatted for both both US letter and A4 formats) in the source tarball in doc/doxygen-doc/.

**NOTE:** If you are building the documentation from a Git clone, you will need to have Doxygen and pdflatex installed -- the documentation will be built during the normal "make" process. The documentation is installed during "make install" to $prefix/share/doc/hwloc/ and your systems default man page tree (under $prefix, of course).

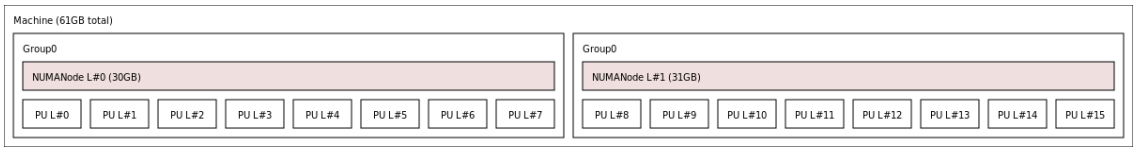
### 2.4.1 Portability

Operating System have varying support for CPU and memory binding, e.g. while some Operating Systems provide interfaces for all kinds of CPU and memory bindings, some others provide only interfaces for a limited number of kinds of CPU and memory binding, and some do not provide any binding interface at all. Hwloc's binding functions would then simply return the ENOSYS error (Function not implemented), meaning that the underlying Operating System does not provide any interface for them. CPU binding and Memory binding provide more information on which hwloc binding functions should be preferred because interfaces for them are usually available on the supported Operating Systems.

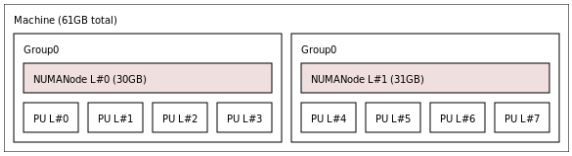
Similarly, the ability of reporting topology information varies from one platform to another. As shown in Command-line Examples, hwloc can obtain information on a wide variety of hardware topologies. However, some platforms and/or operating system versions will only report a subset of this information. For example, on an PPC64-based system with 8 cores (each with 2 hardware threads) running a default 2.6.18-

based kernel from RHEL 5.4, hwloc is only able to glean information about NUMA nodes and processor units (PUs). No information about caches, packages, or cores is available.

Here's the graphical output from lstopo on this platform when Simultaneous Multi-Threading (SMT) is enabled:



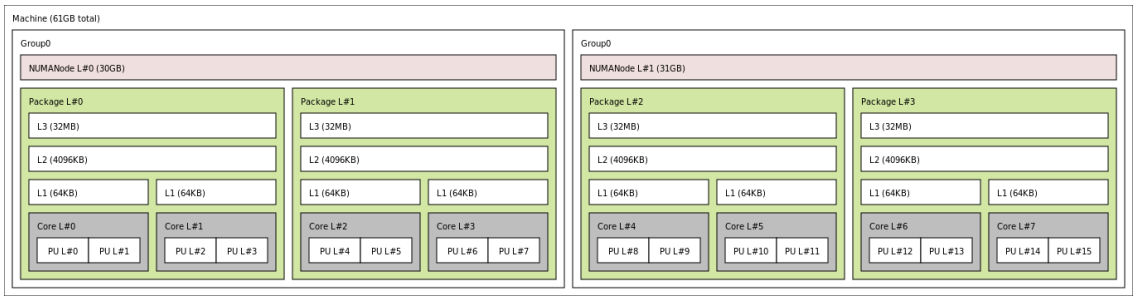

And here's the graphical output from lstopo on this platform when SMT is disabled:




Notice that hwloc only sees half the PUs when SMT is disabled. PU L#6, for example, seems to change location from NUMA node #0 to #1. In reality, no PUs "moved" -- they were simply re-numbered when hwloc only saw half as many (see also Logical index in Indexes and Sets). Hence, PU L#6 in the SMT-disabled picture probably corresponds to PU L#12 in the SMT-enabled picture.

This same "PUs have disappeared" effect can be seen on other platforms -- even platforms / OSs that provide much more information than the above PPC64 system. This is an unfortunate side-effect of how operating systems report information to hwloc.

Note that upgrading the Linux kernel on the same PPC64 system mentioned above to 2.6.34, hwloc is able to discover all the topology information. The following picture shows the entire topology layout when SMT is enabled:




Developers using the hwloc API or XML output for portable applications should therefore be extremely careful to not make any assumptions about the structure of data that is returned. For example, per the above reported PPC topology, it is not safe to assume that PUs will always be descendants of cores.

Additionally, future hardware may insert new topology elements that are not available in this version of hwloc. Long-lived applications that are meant to span multiple different hardware platforms should also be careful about making structure assumptions. For example, a new element may someday exist between a core and a PU.

### 2.4.2 API Example

The following small C example (available in the source tree as "doc/examples/hwloc-hello.c") prints the topology of the machine and performs some thread and memory binding. More examples are available in

the doc/examples/ directory of the source tree.

hwloc provides a `pkg-config` executable to obtain relevant compiler and linker flags. For example, it can be used thusly to compile applications that utilize the hwloc library (assuming GNU Make):

```
CFLAGS += $(shell pkg-config --cflags hwloc)
LDLIBS += $(shell pkg-config --libs hwloc)

hwloc-hello: hwloc-hello.c
        $(CC) hwloc-hello.c $(CFLAGS) -o hwloc-hello $(LDLIBS)
```

On a machine 2 processor packages -- each package of which has two processing cores -- the output from running `hwloc-hello` could be something like the following:

```
shell$ ./hwloc-hello
*** Objects at level 0
Index 0: Machine
*** Objects at level 1
Index 0: Package#0
Index 1: Package#1
*** Objects at level 2
Index 0: Core#0
Index 1: Core#1
Index 2: Core#3
Index 3: Core#2
*** Objects at level 3
Index 0: PU#0
Index 1: PU#1
Index 2: PU#2
Index 3: PU#3
*** Printing overall tree
Machine
  Package#0
    Core#0
      PU#0
    Core#1
      PU#1
  Package#1
    Core#3
      PU#2
    Core#2
      PU#3
*** 2 package(s)
*** Logical processor 0 has 0 caches totaling 0KB
shell$
```

## 2.5 History / Credits

hwloc is the evolution and merger of the libtopology project and the Portable Linux Processor Affinity (PLPA) (https://www.open-mpi.org/projects/plpa/) project. Because of functional and ideological overlap, these two code bases and ideas were merged and released under the name "hwloc" as an Open MPI sub-project.

libtopology was initially developed by the Inria Runtime Team-Project. PLPA was initially developed by the Open MPI development team as a sub-project. Both are now deprecated in favor of hwloc, which is distributed as an Open MPI sub-project.

## 2.6 Further Reading

The documentation chapters include

- Terms and Definitions
- Command-Line Tools
- Environment Variables
- CPU and Memory Binding Overview
- I/O Devices
- Miscellaneous objects
- Object attributes
- Importing and exporting topologies from/to XML files
- Synthetic topologies
- Interoperability With Other Software
- Thread Safety
- Components and plugins
- Embedding hwloc in Other Software
- Frequently Asked Questions
- Upgrading to the hwloc 2.0 API

Make sure to have had a look at those too!

# Chapter 3

# Terms and Definitions

## 3.1 Objects

**Object** Interesting kind of part of the system, such as a Core, a L2Cache, a NUMA memory node, etc. The different types detected by hwloc are detailed in the hwloc_obj_type_t enumeration.

There are four kinds of Objects: Memory (NUMA nodes and Memory-side caches), I/O (Bridges, PCI and OS devices), Misc, and Normal (everything else, including Machine, Package, Die, Core, PU, CPU Caches, etc.). Normal and Memory objects have (non-NULL) CPU sets and nodesets, while I/O and Misc don't.

Objects are topologically sorted by locality (CPU and node sets) into a tree (see Hierarchy, Tree and Levels).

**Processing Unit (or Logical Processor)** The smallest processing element that can be represented by a hwloc object. It may be a single-core processor, a core of a multicore processor, or a single thread in a SMT processor. hwloc's PU acronym stands for Processing Unit.

"Logical processor" should not be confused with "Logical index of a processor".

**Package** A processor Package is the physical package that usually gets inserted into a socket on the motherboard. It is also often called a physical processor or a CPU even if these names bring confusion with respect to cores and processing units. A processor package usually contains multiple cores (and may also be composed of multiple dies). hwloc Package objects were called Sockets up to hwloc 1.10.

**NUMA Node** An object that contains memory that is directly and byte-accessible to the host processors. It is usually close to some cores as specified by its CPU set. Hence it is attached as a memory child of the object that groups those cores together, for instance a Package objects with 4 Core children (see Hierarchy, Tree and Levels).

**Memory-side Cache** A cache in front of a specific memory region (e.g. a range of physical addresses). It caches all accesses to that region without caring about which core issued the request. This is the opposite of usual CPU caches where only accesses from the local cores are cached, without caring about the target memory.

In hwloc, memory-side caches are memory objects placed between their local CPU objects (parent) and the target NUMA node memory (child).

## 3.2 Indexes and Sets

**OS or physical index** The index that the operating system (OS) uses to identify the object. This may be completely arbitrary, non-unique, non-contiguous, not representative of logical proximity, and may depend on the BIOS configuration. That is why hwloc almost never uses them, only in the default lstopo output (`P#x`) and cpuset masks. See also Should I use logical or physical/OS indexes? and how?.

**Logical index** Index to uniquely identify objects of the same type and depth, automatically computed by hwloc according to the topology. It expresses logical proximity in a generic way, i.e. objects which have adjacent logical indexes are adjacent in the topology. That is why hwloc almost always uses it in its API, since it expresses logical proximity. They can be shown (as `L#x`) by `lstopo` thanks to the `-l` option. This index is always linear and in the range [0, num_objs_same_type_same_level-1]. Think of it as "cousin rank." The ordering is based on topology first, and then on OS CPU numbers, so it is stable across everything except firmware CPU renumbering. "Logical index" should not be confused with "Logical processor". A "Logical processor" (which in hwloc we rather call "processing unit" to avoid the confusion) has both a physical index (as chosen arbitrarily by BIOS/OS) and a logical index (as computed according to logical proximity by hwloc). See also Should I use logical or physical/OS indexes? and how?.

**CPU set** The set of logical processors (or processing units) logically included in an object (if it makes sense). They are always expressed using physical logical processor numbers (as announced by the OS). They are implemented as the hwloc_bitmap_t opaque structure. hwloc CPU sets are just masks, they do *not* have any relation with an operating system actual binding notion like Linux' cpusets. I/O and Misc objects do not have CPU sets while all Normal and Memory objects have non-NULL CPU sets.

**Node set** The set of NUMA memory nodes logically included in an object (if it makes sense). They are always expressed using physical node numbers (as announced by the OS). They are implemented with the hwloc_bitmap_t opaque structure. as bitmaps. I/O and Misc objects do not have Node sets while all Normal and Memory objects have non-NULL nodesets.

**Bitmap** A possibly-infinite set of bits used for describing sets of objects such as CPUs (CPU sets) or memory nodes (Node sets). They are implemented with the hwloc_bitmap_t opaque structure.

## 3.3 Hierarchy, Tree and Levels

**Parent object** The object logically containing the current object, for example because its CPU set includes the CPU set of the current object. All objects have a non-NULL parent, except the root of the topology (Machine object).

**Ancestor object** The parent object, or its own parent, and so on.

**Children object(s)** The object (or objects) contained in the current object because their CPU set is included in the CPU set of the current object. Each object may also contain separated lists for Memory, I/O and Misc object children.

**Arity** The number of normal children of an object. There are also specific arities for Memory, I/O and Misc children.

**Sibling objects** Objects in the same children list, which all of them are normal children of the same parent, or all of them are Memory children of the same parent, or I/O children, or Misc. They usually have the same type (and hence are cousins, as well). But they may not if the topology is asymmetric.

**Sibling rank** Index to uniquely identify objects which have the same parent, and is always in the range [0, arity-1] (respectively memory_arity, io_arity or misc_arity for Memory, I/O and Misc children of a parent).

**Cousin objects** Objects of the same type (and depth) as the current object, even if they do not have the same parent.

**Level** Set of objects of the same type and depth. All these objects are cousins.

Memory, I/O and Misc objects also have their own specific levels and (virtual) depth.

**Depth** Nesting level in the object tree, starting from the root object. If the topology is symmetric, the depth of a child is equal to the parent depth plus one, and an object depth is also equal to the number of parent/child links between the root object and the given object. If the topology is asymmetric, the difference between some parent and child depths may be larger than one when some intermediate levels (for instance groups) are missing in only some parts of the machine.

The depth of the Machine object is always 0 since it is always the root of the topology. The depth of PU objects is equal to the number of levels in the topology minus one.

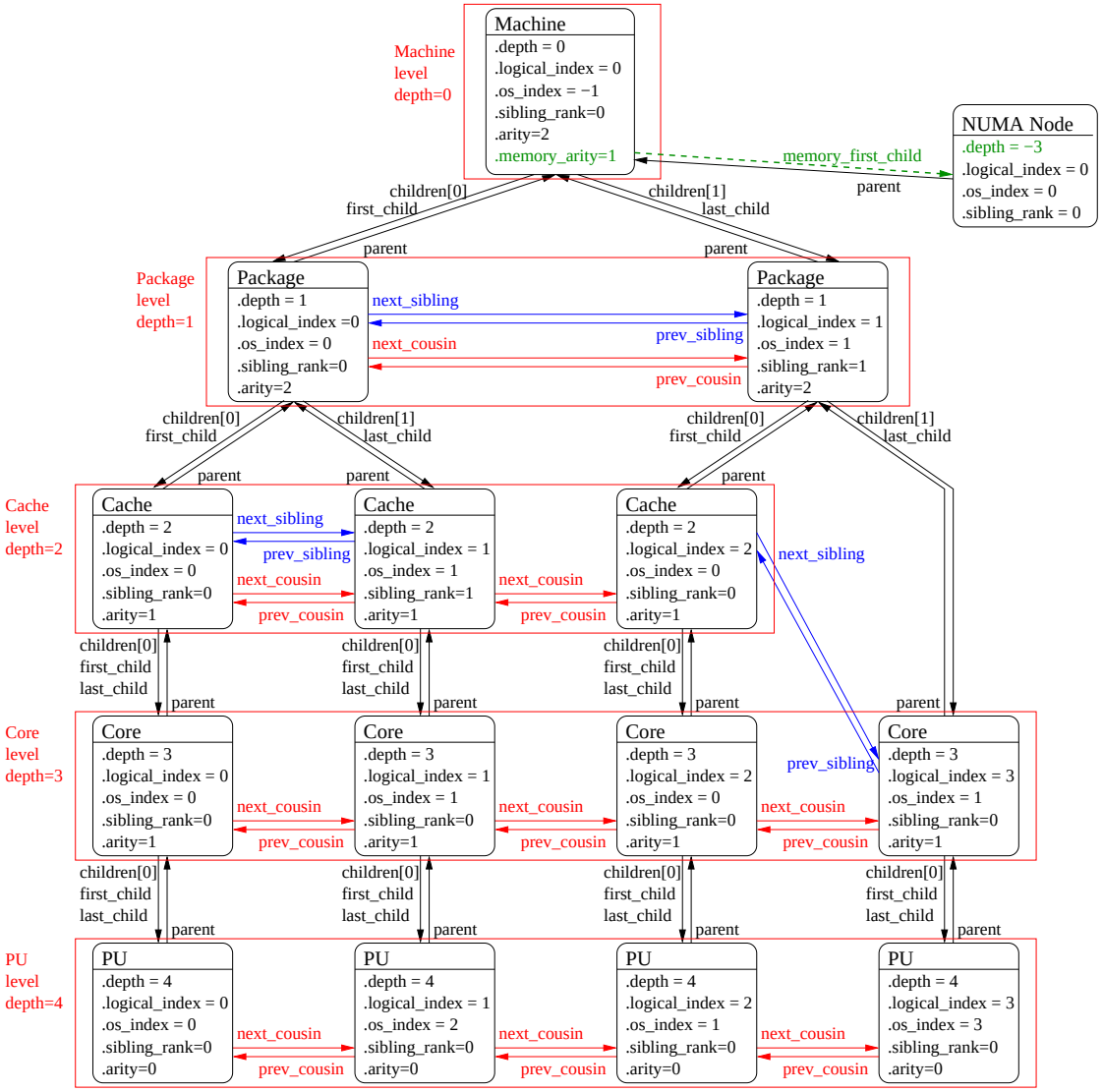
Memory, I/O and Misc objects also have their own specific levels and depth.

The following diagram can help to understand the vocabulary of the relationships by showing the example of a machine with two dual core packages (with no hardware threads); thus, a topology with 5 levels. Each box with rounded corner corresponds to one hwloc_obj_t, containing the values of the different integer fields (depth, logical_index, etc.), and arrows show to which other hwloc_obj_t pointers point to (first_child, parent, etc.).

The topology always starts with a Machine object as root (depth 0) and ends with PU objects at the bottom (depth 4 here).

Objects of the same level (cousins) are listed in red boxes and linked with red arrows. Children of the same parent (siblings) are linked with blue arrows.

The L2 cache of the last core is intentionally missing to show how asymmetric topologies are handled. See What happens if my topology is asymmetric? for more information about such strange topologies.




It should be noted that for PU objects, the logical index -- as computed linearly by hwloc -- is not the same as the OS index.

The NUMA node is on the side because it is not part of the main tree but rather attached to the object that corresponds to its locality (the entire machine here, hence the root object). It is attached as a *Memory* child

(in green) and has a virtual depth (negative). It could also have siblings if there were multiple local NUMA nodes, or cousins if other NUMA nodes were attached somewhere else in the machine.

I/O or Misc objects could be attached in a similar manner.

# Chapter 4

# Command-Line Tools

hwloc comes with an extensive C programming interface and several command line utilities. Each of them is fully documented in its own manual page; the following is a summary of the available command line tools.

## 4.1 lstopo and lstopo-no-graphics

lstopo (also known as hwloc-ls) displays the hierarchical topology map of the current system. The output may be graphical, ascii-art or textual, and can also be exported to numerous file formats such as PDF, PNG, XML, and others. Advanced graphical outputs require the "Cairo" development package (usually `cairo-devel` or `libcairo2-dev`).

lstopo and lstopo-no-graphics accept the same command-line options. However, graphical outputs are only available in lstopo. Textual outputs (those that do not depend on heavy external libraries such as Cairo) are supported in both lstopo and lstopo-no-graphics.

This command can also display the processes currently bound to a part of the machine (via the `\--ps` option).

Note that lstopo can read XML files and/or alternate chroot filesystems and display topological maps representing those systems (e.g., use lstopo to output an XML file on one system, and then use lstopo to read in that XML file and display it on a different system).

## 4.2 hwloc-bind

hwloc-bind binds processes to specific hardware objects through a flexible syntax. A simple example is binding an executable to specific cores (or packages or bitmaps or ...). The hwloc-bind(1) man page provides much more detail on what is possible.

hwloc-bind can also be used to retrieve the current process' binding, or retrieve the last CPU(s) where a process ran, or operate on memory binding.

Just like hwloc-calc, the input locations given to hwloc-bind may be either objects or cpusets (bitmaps as reported by hwloc-calc or hwloc-distrib).

## 4.3 hwloc-calc

hwloc-calc is hwloc's Swiss Army Knife command-line tool for converting things. The input may be either objects or cpusets (bitmaps as reported by another hwloc-calc instance or by hwloc-distrib), that may be combined by addition, intersection or subtraction. The output kinds include:

- a cpuset bitmap: This compact opaque representation of objects is useful for shell scripts etc. It may passed to hwloc command-line tools such as hwloc-calc or hwloc-bind, or to hwloc command-line options such as `lstopo \--restrict`.
- the amount of the equivalent hwloc objects from a specific type, or the list of their indexes. This is useful for iterating over all similar objects (for instance all cores) within a given part of a platform.
- a hierarchical description of objects, for instance a thread index within a core within a package. This gives a better view of the actual location of an object.

Moreover, input and/or output may be use either physical/OS object indexes or as hwloc's logical object indexes. It eases cooperation with external tools such as taskset or numactl by exporting hwloc specifica-

tions into list of processor or NUMA node physical indexes. See also Should I use logical or physical/OS indexes? and how?.

## 4.4 hwloc-info

hwloc-info dumps information about the given objects, as well as all its specific attributes. It is intended to be used with tools such as grep for filtering certain attribute lines. When no object is specified, or when `\--topology` is passed, hwloc-info prints a summary of the topology. When `\--support` is passed, hwloc-info lists the supported features for the topology.

## 4.5 hwloc-distrib

hwloc-distrib generates a set of cpuset bitmaps that are uniformly distributed across the machine for the given number of processes. These strings may be used with hwloc-bind to run processes to maximize their memory bandwidth by properly distributing them across the machine.

## 4.6 hwloc-ps

hwloc-ps is a tool to display the bindings of processes that are currently running on the local machine. By default, hwloc-ps only lists processes that are bound; unbound process (and Linux kernel threads) are not displayed.

## 4.7 hwloc-annotate

hwloc-annotate may modify object (and topology) attributes such as string information (see Custom string infos for details) or Misc children objects. It reads an input topology from a XML file and outputs the annotated topology as another XML file.

## 4.8 hwloc-diff, hwloc-patch and hwloc-compress-dir

hwloc-diff computes the difference between two topologies and outputs it to another XML file.

hwloc-patch reads such a difference file and applies to another topology.

hwloc-compress-dir compresses an entire directory of XML files by using hwloc-diff to save the differences between topologies instead of entire topologies.

## 4.9 hwloc-dump-hwdata

hwloc-dump-hwdata is a Linux and x86-specific tool that dumps (during boot, privileged) some topology and locality information from raw hardware files (SMBIOS and ACPI tables) to human-readable and world-accessible files that the hwloc library will later reuse.

Currently only used on Intel Xeon Phi processor platforms. See Why do I need hwloc-dump-hwdata for memory on Intel Xeon Phi processor?.

See HWLOC_DUMPED_HWDATA_DIR in Environment Variables for details about the location of dumped files.

## 4.10 hwloc-gather-topology and hwloc-gather-cpuid

hwloc-gather-topology is a Linux-specific tool that saves the relevant topology files of the current machine into a tarball (and the corresponding lstopo outputs).

hwloc-gather-cpuid is a x86-specific tool that dumps the result of CPUID instructions on the current machine into a directory.

The output of hwloc-gather-cpuid is included in the tarball saved by hwloc-gather-topology when running on Linux/x86.

These files may be used later (possibly offline) for simulating or debugging a machine without actually running on it.

# Chapter 5

# Environment Variables

The behavior of the hwloc library and tools may be tuned thanks to the following environment variables.

**HWLOC_XMLFILE=/path/to/file.xml** enforces the discovery from the given XML file as if hwloc_topology_set_xml() had been called. This file may have been generated earlier with lstopo file.xml. For convenience, this backend provides empty binding hooks which just return success. To have hwloc still actually call OS-specific hooks, HWLOC_THISSYSTEM should be set 1 in the environment too, to assert that the loaded file is really the underlying system. See also Importing and exporting topologies from/to XML files.

**HWLOC_SYNTHETIC=synthetic_description** enforces the discovery through a synthetic description string as if hwloc_topology_set_synthetic() had been called. For convenience, this backend provides empty binding hooks which just return success. See also Synthetic topologies.

**HWLOC_XML_VERBOSE=1**

**HWLOC_SYNTHETIC_VERBOSE=1** enables verbose messages in the XML or synthetic topology backends. hwloc XML backends (see Importing and exporting topologies from/to XML files) can emit some error messages to the error output stream. Enabling these verbose messages within hwloc can be useful for understanding failures to parse input XML topologies. Similarly, enabling verbose messages in the synthetic topology backend can help understand why the description string is invalid. See also Synthetic topologies.

**HWLOC_THISSYSTEM=1** enforces the return value of hwloc_topology_is_thissystem(), as if HWLOC_TOPOLOGY_FLAG_IS_THISSYSTEM was set with hwloc_topology_set_flags(). It means that it makes hwloc assume that the selected backend provides the topology for the system on which we are running, even if it is not the OS-specific backend but the XML backend for instance. This means making the binding functions actually call the OS-specific system calls and really do binding, while the XML backend would otherwise provide empty hooks just returning success. This can be used for efficiency reasons to first detect the topology once, save it to an XML file, and quickly reload it later through the XML backend, but still having binding functions actually do bind. This also enables support for the variable HWLOC_THISSYSTEM_ALLOWED_RESOURCES.

**HWLOC_THISSYSTEM_ALLOWED_RESOURCES=1** Get the set of allowed resources from the native operating system even if the topology was loaded from XML or synthetic description, as if HWLOC_TOPOLOGY_FLAG_THISSYSTEM_ALLOWED_RESOURCES was set with hwloc_topology_set_flags(). This variable requires the topology to match the current system (see the variable HWLOC_THISSYSTEM). This is useful when the topology is not loaded directly from the local machine (e.g. for performance reason) and it comes with all resources, but the running process is restricted to only a part of the machine (for instance because of Linux Cgroup/Cpuset).

**HWLOC_ALLOW=all** Totally ignore administrative restrictions such as Linux Cgroups and consider all resources (PUs and NUMA nodes) as allowed. This is different from setting HWLOC_TOPOLOGY_FLAG_INCLUDE_DISALLOWED which gathers all resources but marks the unavailable ones as disallowed.

**HWLOC_HIDE_ERRORS=0** enables or disables verbose reporting of errors. The hwloc library may issue warnings to the standard error stream when it detects a problem during topology discovery, for instance if the operating system (or user) gives contradictory topology information. Setting this environment variable to 1 removes the actual displaying of these error messages.

**HWLOC_USE_NUMA_DISTANCES=7** enables or disables the use of NUMA distances. NUMA distances and memory target/initiator information may be used to improve the locality of NUMA nodes, especially CPU-less nodes. Bits in the value of this environment variable enable different features: Bit 0 enables the gathering of NUMA distances from the operating system. Bit 1 further enables the use of NUMA distances to improve the locality of CPU-less nodes. Bit 2 enables the use of target/initiator information.

**HWLOC_GROUPING=1** enables or disables objects grouping based on distances. By default, hwloc uses distance matrices between objects (either read from the OS or given by the user) to find groups of close objects. These groups are described by adding intermediate Group objects in the topology. Setting this environment variable to 0 will disable this grouping. This variable supersedes the obsolete HWLOC_IGNORE_DISTANCES variable.

**HWLOC_GROUPING_ACCURACY=0.05** relaxes distance comparison during grouping. By default, objects may be grouped if their distances form a minimal distance graph. When setting this variable to 0.02, and when HWLOC_DISTANCES_ADD_FLAG_GROUP_INACCURATE is given, these distances do not have to be strictly equal anymore, they may just be equal with a 2% error. If set to `try` instead of a numerical value, hwloc will try to group with perfect accuracy (0, the default), then with 0.01, 0.02, 0.05 and finally 0.1. Numbers given in this environment variable should always use a dot as a decimal mark (for instance 0.01 instead of 0,01).

**HWLOC_GROUPING_VERBOSE=0** enables or disables some verbose messages during grouping. If this variable is set to 1, some debug messages will be displayed during distance-based grouping of objects even if debug was not specific at configure time. This is useful when trying to find an interesting distance grouping accuracy.

**HWLOC_PCI_LOCALITY=<domain/bus> <cpuset>;...**

**HWLOC_PCI_LOCALITY=/path/to/pci/locality/file** changes the locality of I/O devices behing the specified PCI buses. If no I/O locality information is available or if the BIOS reports incorrect information, it is possible to move a I/O device tree (OS and/or PCI devices with optional bridges) near a custom set of processors.

Localities are given either inside the environment variable itself, or in the pointed file. They may be separated either by semi-colons or by line-breaks.

Each locality contains a domain/bus specification (in hexadecimal numbers as usual) followed by a whitespace and a cpuset:

- `0001 <cpuset>` specifies the locality of all buses in PCI domain 0000.
- `0000:0f <cpuset>` specifies only PCI bus 0f in domain 0000.
- `0002:04-0a <cpuset>` specifies a range of buses (from 04 to 0a) within domain 0002.

Domain/bus specifications should usually match entire hierarchies of buses behind a bridge (including primary, secondary and subordinate buses). For instance, if hostbridge 0000:00 is above other bridges/switches with buses 0000:01 to 0000:09, the variable should be HWLOC_PCI_LOCALITY="0000:00-09 <cpuset>". It supersedes the old HWLOC_PCI_0000_00_LOCALCPUS=<cpuset> which only works when hostbridges exist in the topology.

If the variable is defined to empty or invalid, no forced PCI locality is applied but hwloc's internal automatic locality quirks are disabled, which means the exact PCI locality reported by the platform is used.

**HWLOC_X86_TOPOEXT_NUMANODES=0** use AMD topoext CPUID leaf in the x86 backend to detect NUMA nodes. When using the x86 backend, setting this variable to 1 enables the building of NUMA nodes from AMD processor CPUID instructions. However this strategy does not always reflect BIOS configuration such as NUMA interleaving. And node indexes may be different from those of the operating system. Hence this should only be used when OS backends are wrong and the user is sure that CPUID returns correct NUMA information.

**HWLOC_KEEP_NVIDIA_GPU_NUMA_NODES=0** show or hide NUMA nodes that correspond to NVIDIA GPU memory. By default they are ignored to avoid interleaved memory being allocated on GPU by mistake. Setting this environment variable to 1 exposes these NUMA nodes. They may be recognized by the *GPUMemory* subtype. They also have a *PCIBusID* info attribute to identify the corresponding GPU.

**HWLOC_KNL_MSCACHE_L3=0** Expose the KNL MCDRAM in cache mode as a Memory-side Cache instead of a L3. hwloc releases prior to 2.1 exposed the MCDRAM cache as a CPU-side L3 cache. Now that Memory-side caches are supported by hwloc, it is still exposed as a L3 by default to avoid breaking existing applications. Setting this environment variable to 1 will expose it as a proper Memory-side cache.

**HWLOC_ANNOTATE_GLOBAL_COMPONENTS=0** Allow components to annotate the topology even if they are usually excluded by global components by default. Setting this variable to 1 and also setting `HWLOC_COMPONENTS=xml,pci,stop` enables the addition of PCI vendor and model info attributes to a XML topology that was generated without those names (if pciaccess was missing).

**HWLOC_FSROOT=/path/to/linux/filesystem-root/** switches to reading the topology from the specified Linux filesystem root instead of the main file-system root. This directory may have been saved previously from another machine with `hwloc-gather-topology`.

One should likely also set `HWLOC_COMPONENTS=linux,stop` so that non-Linux backends are disabled (the `-i` option of command-line tools takes care of both).

Not using the main file-system root causes hwloc_topology_is_thissystem() to return 0. For convenience, this backend provides empty binding hooks which just return success. To have hwloc still actually call OS-specific hooks, HWLOC_THISSYSTEM should be set 1 in the environment too, to assert that the loaded file is really the underlying system.

**HWLOC_CPUID_PATH=/path/to/cpuid/** forces the x86 backend to read dumped CPUIDs from the given directory instead of executing actual x86 CPUID instructions. This directory may have been saved previously from another machine with `hwloc-gather-cpuid`.

One should likely also set `HWLOC_COMPONENTS=x86,stop` so that non-x86 backends are disabled (the `-i` option of command-line tools takes care of both).

It causes hwloc_topology_is_thissystem() to return 0. For convenience, this backend provides empty binding hooks which just return success. To have hwloc still actually call OS-specific hooks, HWLOC_THISSYSTEM should be set 1 in the environment too, to assert that the loaded CPUID dump is really the underlying system.

**HWLOC_DUMPED_HWDATA_DIR=/path/to/dumped/files/** loads files dumped by `hwloc-dump-hwdata` (on Linux) from the given directory. The default dump/load directory is configured during build based on \--runstatedir, \--localstatedir, and \--prefix options. It usually points to `/var/run/hwloc/` in Linux distribution packages, but it may also point to `$prefix/var/run/hwloc/` when manually installing and only specifying \--prefix.

**HWLOC_COMPONENTS=list,of,components** forces a list of components to enable or disable. Enable or disable the given comma-separated list of components (if they do not conflict with each other). Component names prefixed with - are disabled (a single phase may also be disabled).

Once the end of the list is reached, hwloc falls back to enabling the remaining components (sorted by priority) that do not conflict with the already enabled ones, and unless explicitly disabled in the list. If `stop` is met, the enabling loop immediately stops, no more component is enabled.

If `xml` or `synthetic` components are selected, the corresponding XML filename or synthetic description string should be pass in `HWLOC_XMLFILE` or `HWLOC_SYNTHETIC` respectively.

Since this variable is the low-level and more generic way to select components, it takes precedence over environment variables for selecting components.

If the variable is set to an empty string (or set to a single comma), no specific component is loaded first, all components are loaded in priority order.

See Selecting which components to use for details.

**HWLOC_COMPONENTS_VERBOSE=1** displays verbose information about components. Display messages when components are registered or enabled. This is the recommended way to list the available components with their priority (all of them are *registered* at startup).

**HWLOC_PLUGINS_PATH=/path/to/hwloc/plugins/:...** changes the default search directory for plugins. By default, `$libdir/hwloc` is used. The variable may contain several colon-separated directories.

**HWLOC_PLUGINS_VERBOSE=1** displays verbose information about plugins. List which directories are scanned, which files are loaded, and which components are successfully loaded.

**HWLOC_PLUGINS_BLACKLIST=filename1,filename2,...** prevents plugins from being loaded if their filename (without path) is listed. Plugin filenames may be found in verbose messages outputted when HWLOC_PLUGINS_VERBOSE=1.

**HWLOC_DEBUG_VERBOSE=0** disables all verbose messages that are enabled by default when `--enable-debug` is passed to configure.

# Chapter 6

# CPU and Memory Binding Overview

Some operating systems do not systematically provide separate functions for CPU and memory binding. This means that CPU binding functions may have have effects on the memory binding policy. Likewise, changing the memory binding policy may change the CPU binding of the current thread. This is often not a problem for applications, so by default hwloc will make use of these functions when they provide better binding support.

If the application does not want the CPU binding to change when changing the memory policy, it needs to use the HWLOC_MEMBIND_NOCPUBIND flag to prevent hwloc from using OS functions which would change the CPU binding. Additionally, HWLOC_CPUBIND_NOMEMBIND can be passed to CPU binding function to prevent hwloc from using OS functions would change the memory binding policy. Of course, using these flags will reduce hwloc's overall support for binding, so their use is discouraged.

One can avoid using these flags but still closely control both memory and CPU binding by allocating memory, touching each page in the allocated memory, and then changing the CPU binding. The already-really-allocated memory will then be "locked" to physical memory and will not be migrated. Thus, even if the memory binding policy gets changed by the CPU binding order, the already-allocated memory will not change with it. When binding and allocating further memory, the CPU binding should be performed again in case the memory binding altered the previously-selected CPU binding.

Not all operating systems support the notion of a "current" memory binding policy for the current process, but such operating systems often still provide a way to allocate data on a given node set. Conversely, some operating systems support the notion of a "current" memory binding policy and do not permit allocating data on a specific node set without changing the current policy and allocate the data. To provide the most powerful coverage of these facilities, hwloc provides:

- functions that set/get the current memory binding policies (if supported): hwloc_set/get_membind() and hwloc_set/get_proc_membind()
- a function that allocates memory bound to specific node set without changing the current memory binding policy (if supported): hwloc_alloc_membind().
- a helper which, if needed, changes the current memory binding policy of the process in order to obtain memory binding: hwloc_alloc_membind_policy().

An application can thus use the two first sets of functions if it wants to manage separately the global process binding policy and directed allocation, or use the third set of functions if it does not care about the process memory binding policy.

See CPU binding and Memory binding for hwloc's API functions regarding CPU and memory binding, respectively. There are some examples under doc/examples/ in the source tree.

# Chapter 7

# I/O Devices

hwloc usually manipulates processing units and memory but it can also discover I/O devices and report their locality as well. This is useful for placing I/O intensive applications on cores near the I/O devices they use, or for gathering information about all platform components.

## 7.1 Enabling and requirements

I/O discovery is disabled by default (except in lstopo) for performance reasons. It can be enabled by changing the filtering of I/O object types to `HWLOC_TYPE_FILTER_KEEP_IMPORTANT` or `HWLOC_TYPE_FILTER_KEEP_ALL` before loading the topology, for instance with `hwloc_topology_set_io_types_filter()`.

Note that I/O discovery requires significant help from the operating system. The pciaccess library (the development package is usually `libpciaccess-devel` or `libpciaccess-dev`) is needed to fully detect PCI devices and bridges/switches. On Linux, PCI discovery may still be performed even if `libpciaccess` cannot be used. But it misses PCI device names. Moreover, some operating systems require privileges for probing PCI devices, see Does hwloc require privileged access? for details.

The actual locality of I/O devices is only currently detected on Linux. Other operating system will just report I/O devices as being attached to the topology root object.

## 7.2 I/O objects

When I/O discovery is enabled and supported, some additional objects are added to the topology. The corresponding I/O object types are:

- `HWLOC_OBJ_OS_DEVICE` describes an operating-system-specific handle such as the *sda* drive or the *eth0* network interface. See OS devices.
- `HWLOC_OBJ_PCI_DEVICE` and `HWLOC_OBJ_BRIDGE` build up a PCI hierarchy made of bridges (that may be actually be switches) and devices. See PCI devices and bridges.

Any of these types may be filtered individually with `hwloc_topology_set_type_filter()`.

hwloc tries to attach these new objects to normal objects (usually NUMA nodes) to match their actual physical location. For instance, if a I/O hub (or root complex) is physically connected to a package, the corresponding hwloc bridge object (and its PCI bridges and devices children) is inserted as a child of the corresponding hwloc Package object. **These children are not in the normal children list but rather in the I/O-specific children list.**

I/O objects also have neither CPU sets nor node sets (NULL pointers) because they are not directly usable by the user applications for binding. Moreover I/O hierarchies may be highly complex (asymmetric trees of bridges). So I/O objects are placed in specific levels with custom depths. Their lists may still be traversed with regular helpers such as hwloc_get_next_obj_by_type(). However, hwloc offers some dedicated helpers such as hwloc_get_next_pcidev() and hwloc_get_next_osdev() for convenience (see Finding I/O objects).

## 7.3 OS devices

Although each PCI device is uniquely identified by its bus ID (e.g. 0000:01:02.3), a user-space application can hardly find out which PCI device it is actually using. Applications rather use software handles (such as

the *eth0* network interface, the *sda* hard drive, or the *mlx4_0* OpenFabrics HCA). Therefore hwloc tries to add software devices (HWLOC_OBJ_OS_DEVICE, also known as OS devices).

OS devices may be attached below PCI devices, but they may also be attached directly to normal objects. Indeed some OS devices are not related to PCI. For instance, NVDIMM block devices (such as *pmem0s* on Linux) are directly attached near their NUMA node (I/O child of the parent whose memory child is the NUMA node). Also, if hwloc could not discover PCI for some reason, PCI-related OS devices may also be attached directly to normal objects.

hwloc first tries to discover OS devices from the operating system, e.g. *eth0*, *sda* or *mlx4_0*. However, this ability is currently only available on Linux for some classes of devices.

hwloc then tries to discover software devices through additional I/O components using external libraries. For instance proprietary graphics drivers do not expose any named OS device, but hwloc may still create one OS object per software handle when supported. For instance the `opencl` and `cuda` components may add some *opencl0d0* and *cuda0* OS device objects.

Here is a list of OS device objects commonly created by hwloc components when I/O discovery is enabled and supported.

- Hard disks or non-volatile memory devices (HWLOC_OBJ_OSDEV_BLOCK)
  - *sda* or *dax2.0* (Linux component)
- Network interfaces (HWLOC_OBJ_OSDEV_NETWORK)
  - *eth0*, *wlan0*, *ib0* (Linux component)
- OpenFabrics (InfiniBand, Omni-Path, usNIC, etc) HCAs (HWLOC_OBJ_OSDEV_-OPENFABRICS)
  - *mlx5_0*, *hfi1_0*, *qib0*, *usnic_0* (Linux component)
- GPUs (HWLOC_OBJ_OSDEV_GPU)
  - *nvml0* for the first NVML device (NVML component, using the NVIDIA Management Library)
  - *:0.0* for the first display (GL component, using the NV-CONTROL X extension library, NVCtrl)
- Co-Processors (HWLOC_OBJ_OSDEV_COPROC)
  - *opencl0d0* for the first device of the first OpenCL platform, *opencl1d3* for the fourth device of the second OpenCL platform (OpenCL component)
  - *cuda0* for the first NVIDIA CUDA device (CUDA component, using the NVIDIA CUDA Library)
  - DMA engine channel (HWLOC_OBJ_OSDEV_DMA)
    - *dma0chan0* (Linux component) when all OS devices are enabled (HWLOC_TYPE_-FILTER_KEEP_ALL)

Note that some PCI devices may contain multiple software devices (see the example below).

See also Interoperability With Other Software for managing these devices without considering them as hwloc objects.

## 7.4 PCI devices and bridges

A PCI hierarchy is usually organized as follows: A hostbridge object ( HWLOC_OBJ_BRIDGE object with upstream type *Host* and downstream type *PCI*) is attached below a normal object (usually the entire machine or a NUMA node). There may be multiple hostbridges in the machine, attached to different places, but all PCI devices are below one of them (unless the Bridge object type is filtered-out).

Each hostbridge contains one or several children, either other bridges (usually PCI to PCI switches) or PCI devices (HWLOC_OBJ_PCI_DEVICE). The number of bridges between the hostbridge and a PCI device depends on the machine.

## 7.5 Consulting I/O devices and binding

I/O devices may be consulted by traversing the topology manually (with usual routines such as hwloc_get_obj_by_type()) or by using dedicated helpers (such as hwloc_get_pcidev_by_busid(), see Finding I/O objects).

I/O objects do not actually contain any locality information because their CPU sets and node sets are NULL. Their locality must be retrieved by walking up the object tree (through the `parent` link) until an non-I/O object is found (see hwloc_get_non_io_ancestor_obj()). This normal object should have non-NULL CPU sets and node sets which describe the processing units and memory that are immediately close to the I/O device. For instance the path from a OS device to its locality may go across a PCI device parent, one or several bridges, up to a Package node with the same locality.

Command-line tools are also aware of I/O devices. lstopo displays the interesting ones by default (passing `\--no-io` disables it).

hwloc-calc and hwloc-bind may manipulate I/O devices specified by PCI bus ID or by OS device name.

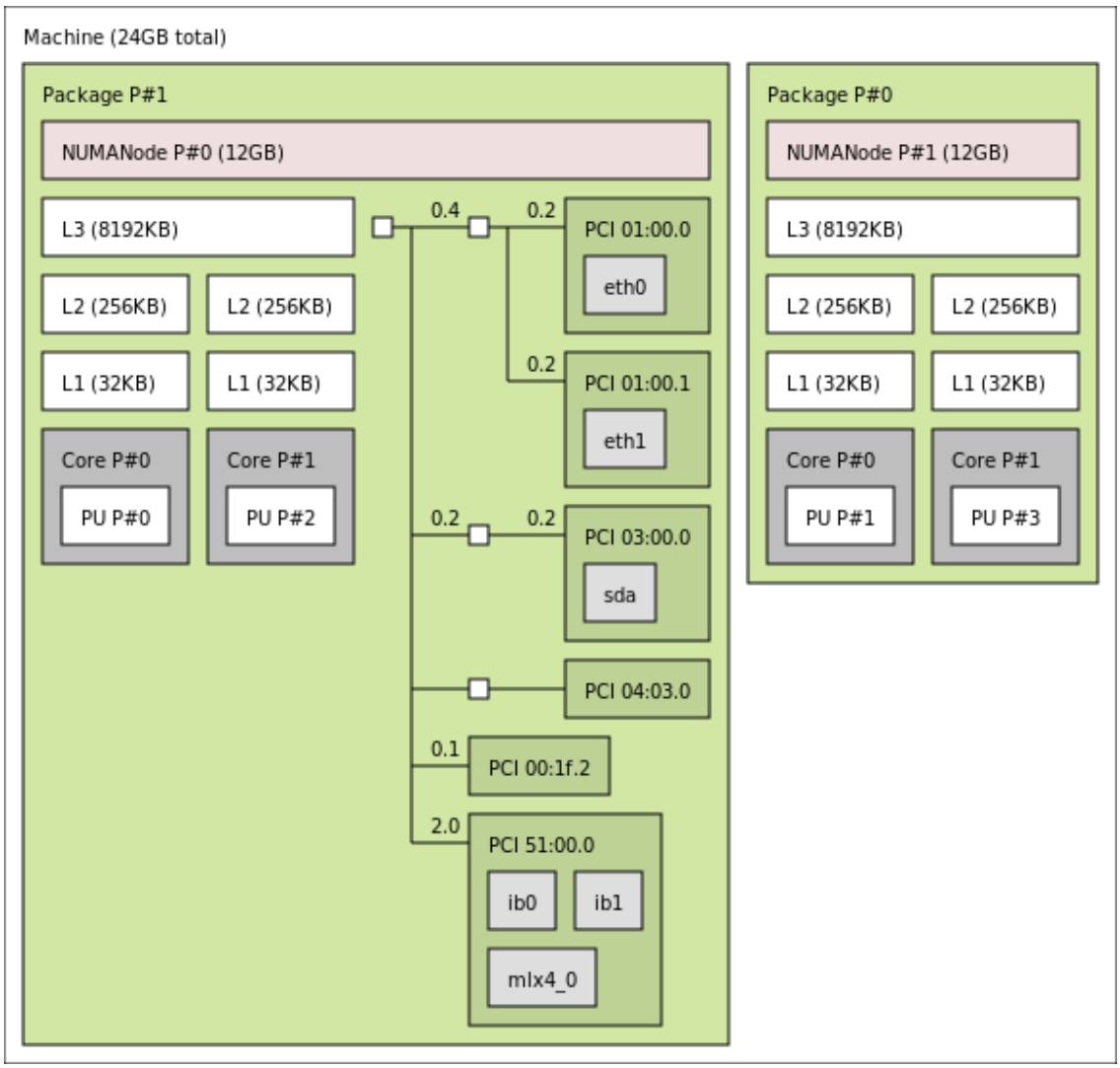
- `pci=0000:02:03.0` is replaced by the set of CPUs that are close to the PCI device whose bus ID is given.

- `os=eth0` is replaced by CPUs that are close to the I/O device whose software handle is called `eth0`.

This enables easy binding of I/O-intensive applications near the device they use.

## 7.6 Examples

The following picture shows a dual-package dual-core host whose PCI bus is connected to the first package and NUMA node.




Six interesting PCI devices were discovered. However, hwloc found some corresponding software devices (*eth0*, *eth1*, *sda*, *mlx4_0*, *ib0*, and *ib1*) for only four of these physical devices. The other ones (*PCI 102b:0532* and *PCI 8086:3a20*) are an unused IDE controller (no disk attached) and a graphic card (no corresponding software device reported to the user by the operating system).

On the contrary, it should be noted that three different software devices were found for the last PCI device (*PCI 15b3:634a*). Indeed this OpenFabrics HCA PCI device object contains one one OpenFabrics software device (*mlx4_0*) and two virtual network interface software devices (*ib0* and *ib1*).

Here is the corresponding textual output:

```
Machine (24GB total)
  Package L#0
    NUMANode L#0 (P#0 12GB)
    L3 L#0 (8192KB)
      L2 L#0 (256KB) + L1 L#0 (32KB) + Core L#0 + PU L#0 (P#0)
      L2 L#1 (256KB) + L1 L#1 (32KB) + Core L#1 + PU L#1 (P#2)
    HostBridge
      PCIBridge
        PCI 01:00.0 (Ethernet)
          Net "eth0"
        PCI 01:00.1 (Ethernet)
```

```
        Net "eth1"
    PCIBridge
      PCI 03:00.0 (RAID)
        Block "sda"
    PCIBridge
      PCI 04:03.0 (VGA)
    PCI 00:1f.2 (IDE)
    PCI 51:00.0 (InfiniBand)
      Net "ib0"
      Net "ib1"
      Net "mlx4_0"
Package L#1
  NUMANode L#1 (P#1 12GB)
  L3 L#1 (8192KB)
    L2 L#2 (256KB) + L1 L#2 (32KB) + Core L#2 + PU L#2 (P#1)
    L2 L#3 (256KB) + L1 L#3 (32KB) + Core L#3 + PU L#3 (P#3)
```

# Chapter 8

# Miscellaneous objects

hwloc topologies may be annotated with Misc objects (of type HWLOC_OBJ_MISC) either automatically or by the user. This is an flexible way to annotate topologies with large sets of information since Misc objects may be inserted anywhere in the topology (to annotate specific objects or parts of the topology), even below other Misc objects, and each of them may contain multiple attributes (see also How do I annotate the topology with private notes?).

These Misc objects may have a subtype field to replace Misc with something else in the lstopo output.

## 8.1 Misc objects added by hwloc

hwloc only uses Misc objects when other object types are not sufficient, and when the Misc object type is not filtered-out anymore. This currently includes:

- Memory modules (DIMMs), on Linux when privileged and when dmi-sysfs is supported by the kernel. These objects have a subtype field of value MemoryModule. They are currently always attached to the root object. Their attributes describe the DIMM vendor, model, etc. lstopo -v displays them as:

```
Misc(MemoryModule) (P#1 DeviceLocation="Bottom-Slot 2(right)" BankLocation="BANK
      2" Vendor=Elpida SerialNumber=21733667 AssetTag=9876543210 PartNumber="EBJ81UG8EF
      U0-GN-F ")
```

- Displaying process binding in lstopo \--top. These objects have a subtype field of value Process and a name attribute made of their PID and program name. They are attached below the object they are bound to. The textual lstopo displays them as:

```
PU L#0 (P#0)
  Misc(Process) 4445 myprogram
```

## 8.2 Annotating topologies with Misc objects

The user may annotate hwloc topologies with its own Misc objects. This can be achieved with hwloc_topology_insert_misc_object() as well as hwloc-annotate command-line tool.

# Chapter 9

# Object attributes

## 9.1 Normal attributes

hwloc objects have many generic attributes in the hwloc_obj structure, for instance their `logical_index` or `os_index` (see Should I use logical or physical/OS indexes? and how?), `depth` or `name`.

The kind of object is first described by the `obj->type` generic attribute (an integer). OS devices also have a specific `obj->attr->osdev.type` integer for distinguishing between NICs, GPUs, etc. Objects may also have an optional `obj->subtype` pointing to a better description string. For instance subtype is useful to say what Group objects are actually made of (e.g. *Book* for Linux S/390 books). It may also specify that a Block OS device is a *Disk*, or that a CoProcessor OS device is a *CUDA* device. This subtype is displayed by lstopo either in place or after the main `obj->type` attribute. NUMA nodes that correspond GPU memory may also have *GPUMemory* as subtype.

Each object also contains an `attr` field that, if non NULL, points to a union hwloc_obj_attr_u of type-specific attribute structures. For instance, a L2Cache object `obj` contains cache-specific information in `obj->attr->cache`, such as its size and associativity, cache type. See hwloc_obj_attr_u for details.

## 9.2 Custom string infos

Aside os these generic attribute fields, hwloc annotates many objects with string attributes that are made of a key and a value. Each object contains a list of such pairs that may be consulted manually (looking at the object `infos` array field) or using the hwloc_obj_get_info_by_name(). The user may additionally add new key-value pairs to any object using hwloc_obj_add_info() or the hwloc-annotate program.

Here is a non-exhaustive list of attributes that may be automatically added by hwloc. Note that these attributes heavily depend on the ability of the operating system to report them. Many of them will therefore be missing on some OS.

### 9.2.1 Hardware Platform Information

These info attributes are attached to the root object (Machine).

**PlatformName, PlatformModel, PlatformVendor, PlatformBoardID, PlatformRevision,**

**SystemVersionRegister, ProcessorVersionRegister (Machine)** Some POWER/PowerPC-specific attributes describing the platform and processor. Currently only available on Linux. Usually added to Package objects, but can be in Machine instead if hwloc failed to discover any package.

**DMIBoardVendor, DMIBoardName, etc.** DMI hardware information such as the motherboard and chassis models and vendors, the BIOS revision, etc., as reported by Linux under `/sys/class/dmi/id/`.

**MemoryMode, ClusterMode** Intel Xeon Phi processor configuration modes. Available if hwloc-dump-hwdata was used (see Why do I need hwloc-dump-hwdata for memory on Intel Xeon Phi processor?) or if hwloc managed to guess them from the NUMA configuration.

The memory mode may be *Cache*, *Flat*, *Hybrid50* (half the MCDRAM is used as a cache) or *Hybrid25* (25% of MCDRAM as cache). The cluster mode may be *Quadrant*, *Hemisphere*, *All2All*, *SNC2* or *SNC4*. See doc/examples/get-knl-modes.c in the source directory for an example of retrieving these attributes.

### 9.2.2 Operating System Information

These info attributes are attached to the root object (Machine).

**OSName, OSRelease, OSVersion, HostName, Architecture** The operating system name, release, version, the hostname and the architecture name, as reported by the Unix `uname` command.

**LinuxCgroup** The name the Linux control group where the calling process is placed.

### 9.2.3 hwloc Information

Unless specified, these info attributes are attached to the root object (Machine).

**Backend (topology root, or specific object added by that backend)** The name of the hwloc backend/-component that filled the topology. If several components were combined, multiple Backend keys may exist, with different values, for instance `x86` and `Linux` in the root object and `CUDA` in CUDA OS device objects.

**SyntheticDescription** The description string that was given to hwloc to build this synthetic topology.

**hwlocVersion** The version number of the hwloc library that was used to generate the topology. If the topology was loaded from XML, this is not the hwloc version that loaded it, but rather the first hwloc instance that exported the topology to XML earlier.

**ProcessName** The name of the process that contains the hwloc library that was used to generate the topology. If the topology was from XML, this is not the hwloc process that loaded it, but rather the first process that exported the topology to XML earlier.

### 9.2.4 CPU Information

These info attributes are attached to Package objects, or to the root object (Machine) if package locality information is missing.

**CPUModel** The processor model name.

**CPUVendor, CPUModelNumber, CPUFamilyNumber, CPUStepping** The processor vendor name, model number, family number, and stepping number. Currently available for x86 and Xeon Phi processors on most systems, and for ia64 processors on Linux (except CPUStepping).

**CPURevision** A POWER/PowerPC-specific general processor revision number, currently only available on Linux.

**CPUType** A Solaris-specific general processor type name, such as "i86pc".

### 9.2.5 OS Device Information

These info attributes are attached to OS device objects specified in parentheses.

**Vendor, Model, Revision, SerialNumber, Size, SectorSize (Block OS devices)** The vendor and model names, revision, serial number, size (in kB) and SectorSize (in bytes).

**LinuxDeviceID (Block OS devices)** The major/minor device number such as 8:0 of Linux device.

**GPUVendor, GPUModel (GPU or Co-Processor OS devices)** The vendor and model names of the GPU device.

**OpenCLDeviceType, OpenCLPlatformIndex,**

**OpenCLPlatformName, OpenCLPlatformDeviceIndex (OpenCL OS devices)** The type of OpenCL device, the OpenCL platform index and name, and the index of the device within the platform.

**OpenCLComputeUnits, OpenCLGlobalMemorySize (OpenCL OS devices)** The number of compute units and global memory size (in kB) of an OpenCL device.

**NVIDIAUUID, NVIDIASerial (NVML GPU OS devices)** The UUID and Serial of NVIDIA GPUs.

**CUDAMultiProcessors, CUDACoresPerMP,**

**CUDAGlobalMemorySize, CUDAL2CacheSize, CUDASharedMemorySizePerMP (CUDA OS devices)** The number of shared multiprocessors, the number of cores per multiprocessor, the global memory size, the (global) L2 cache size, and size of the shared memory in each multiprocessor of a CUDA device. Sizes are in kB.

**Address, Port (Network interface OS devices)** The MAC address and the port number of a software network interface, such as `eth4` on Linux.

**NodeGUID, SysImageGUID, Port1State, Port2LID, Port2LMC, Port3GID1 (OpenFabrics OS devices)** The node GUID and GUID mask, the state of a port #1 (value is 4 when active), the LID and LID mask count of port #2, and GID #1 of port #3.

### 9.2.6 Other Object-specific Information

These info attributes are attached to objects specified in parentheses.

**DAXDevice (NUMA Nodes)** The name of the Linux DAX device that was used to expose a non-volatile memory region as a volatile NUMA node.

**PCIBusID (GPUMemory NUMA Nodes)** The PCI bus ID of the GPU whose memory is exposed in this NUMA node.

**Inclusive (Caches)** The inclusiveness of a cache (1 if inclusive, 0 otherwise). Currently only available on x86 processors.

**SolarisProcessorGroup (Group)** The Solaris kstat processor group name that was used to build this Group object.

**PCIVendor, PCIDevice (PCI devices and bridges)** The vendor and device names of the PCI device.

**PCISlot (PCI devices or Bridges)** The name/number of the physical slot where the device is plugged. If the physical device contains PCI bridges above the actual PCI device, the attribute may be attached to the highest bridge (i.e. the first object that actually appears below the physical slot).

**Vendor, AssetTag, PartNumber, DeviceLocation, BankLocation (MemoryModule Misc objects)** Information about memory modules (DIMMs) extracted from SMBIOS.

### 9.2.7 User-Given Information

Here is a non-exhaustive list of user-provided info attributes that have a special meaning:

**lstopoStyle** Enforces the style of an object (background and text colors) in the graphical output of lstopo. See CUSTOM COLORS in the lstopo(1) manpage for details.

# Chapter 10

# Importing and exporting topologies from/to XML files

hwloc offers the ability to export topologies to XML files and reload them later. This is for instance useful for loading topologies faster (see I do not want hwloc to rediscover my enormous machine topology every time I rerun a process), manipulating other nodes' topology, or avoiding the need for privileged processes (see Does hwloc require privileged access?).

Topologies may be exported to XML files thanks to hwloc_topology_export_xml(), or to a XML memory buffer with hwloc_topology_export_xmlbuffer(). The lstopo program can also serve as a XML topology export tool.

XML topologies may then be reloaded later with hwloc_topology_set_xml() and hwloc_topology_set_xmlbuffer(). The HWLOC_XMLFILE environment variable also tells hwloc to load the topology from the given XML file (see Environment Variables).

**Note:**

Loading XML topologies disables binding because the loaded topology may not correspond to the physical machine that loads it. This behavior may be reverted by asserting that loaded file really matches the underlying system with the HWLOC_THISSYSTEM environment variable or the HWLOC_TOPOLOGY_FLAG_IS_THISSYSTEM topology flag.
The topology flag HWLOC_TOPOLOGY_FLAG_THISSYSTEM_ALLOWED_RESOURCES may be used to load a XML topology that contains the entire machine and restrict it to the part that is actually available to the current process (e.g. when Linux Cgroup/Cpuset are used to restrict the set of resources).
hwloc also offers the ability to export/import Topology differences.
XML topology files are not localized. They use a dot as a decimal separator. Therefore any exported topology can be reloaded on any other machine without requiring to change the locale.
XML exports contain all details about the platform. It means that two very similar nodes still have different XML exports (e.g. some serial numbers or MAC addresses are different). If a less precise exporting/importing is required, one may want to look at Synthetic topologies instead.

## 10.1 libxml2 and minimalistic XML backends

hwloc offers two backends for importing/exporting XML.

First, it can use the libxml2 library for importing/exporting XML files. It features full XML support, for instance when those files have to be manipulated by non-hwloc software (e.g. a XSLT parser). The libxml2 backend is enabled by default if libxml2 development headers are available (the relevant development package is usually `libxml2-devel` or `libxml2-dev`).

If libxml2 is not available at configure time, or if `\--disable-libxml2` is passed, hwloc falls back to a custom backend. Contrary to the aforementioned full XML backend with libxml2, this minimalistic XML backend cannot be guaranteed to work with external programs. It should only be assumed to be compatible with the same hwloc release (even if using the libxml2 backend). Its advantage is, however, to always be available without requiring any external dependency.

If libxml2 is available but the core hwloc library should not directly depend on it, the libxml2 support may be built as a dynamicall-loaded plugin. One should pass `\--enable-plugins` to enable plugin support (when supported) and build as plugins all component that support it. Or pass `\--enable-plugins=xml_libxml` to only build this libxml2 support as a plugin.

## 10.2 XML import error management

Importing XML files can fail at least because of file access errors, invalid XML syntax, non-hwloc-valid XML contents, or incompatibilities between hwloc releases (see Are XML topology files compatible be-

tween hwloc releases?).

Both backend cannot detect all these errors when the input XML file or buffer is selected (when hwloc_-topology_set_xml() or hwloc_topology_set_xmlbuffer() is called). Some errors such non-hwloc-valid contents can only be detected later when loading the topology with hwloc_topology_load().

It is therefore strongly recommended to check the return value of both hwloc_topology_set_xml() (or hwloc_topology_set_xmlbuffer()) and hwloc_topology_load() to handle all these errors.

# Chapter 11

# Synthetic topologies

hwloc may load fake or remote topologies so as to consult them without having the underlying hardware available. Aside from loading XML topologies, hwloc also enables the building of *synthetic* topologies that are described by a single string listing the arity of each levels.

For instance, lstopo may create a topology made of 2 packages, containing a single NUMA node and a L2 cache above two single-threaded cores:

```
$ lstopo -i "pack:2 node:1 l2:1 core:2 pu:1" -
Machine (2048MB)
  Package L#0
    NUMANode L#0 (P#0 1024MB)
    L2 L#0 (4096KB)
      Core L#0 + PU L#0 (P#0)
      Core L#1 + PU L#1 (P#1)
  Package L#1
    NUMANode L#1 (P#1 1024MB)
    L2 L#1 (4096KB)
      Core L#2 + PU L#2 (P#2)
      Core L#3 + PU L#3 (P#3)
```

Replacing - with `file.xml` in this command line will export this topology to XML as usual.

**Note:**

Synthetic topologies offer a very basic way to export a topology and reimport it on another machine. It is a lot less precise than XML but may still be enough when only the hierarchy of resources matters.

## 11.1 Synthetic description string

Each item in the description string gives the type of the level and the number of such children under each object of the previous level. That is why the above topology contains 4 cores (2 cores times 2 nodes).

These type names must be written as `numanode`, `package`, `core`, `l2u`, `l1i`, `pu`, `group` (hwloc_obj_type_sscanf() is used for parsing the type names). They do not need to be written case-sensitively, nor entirely (as long as there is no ambiguity, 2 characters such as `ma` select a Machine level). Note that I/O and Misc objects are not available.

Instead of specifying the type of each level, it is possible to just specify the arities and let hwloc choose all types according to usual topologies. The following examples are therefore equivalent:

```
$ lstopo -i "2 3 4 5 6"
$ lstopo -i "Package:2 NUMANode:3 L2Cache:4 Core:5 PU:6"
```

NUMA nodes are handled in a special way since they are not part of the main CPU hierarchy but rather attached below it as memory children. Thus, `NUMANode:3` actually means `Group:3` where one NUMA node is attached below each group. These groups are merged back into the parent when possible (typically when a single NUMA node is requested below each parent).

It is also possible the explicitly attach NUMA nodes to specific levels. For instance, a topology similar to a Intel Xeon Phi processor (with 2 NUMA nodes per 16-core group) may be created with:

```
$ lstopo -i "package:1 group:4 [numa] [numa] core:16 pu:4"
```

The root object does not appear in the synthetic description string since it is always a Machine object. Therefore the Machine type is disallowed in the description as well.

A NUMA level (with a single NUMA node) is automatically added if needed.

Each item may be followed parentheses containing a list of space-separated attributes. For instance:

- `L2iCache:2(size=32kB)` specifies 2 children of 32kB level-2 instruction caches. The size may be specified in bytes (without any unit suffix) or as TB, GB, MB or kB.
- `NUMANode:3(memory=16MB)` specifies 3 NUMA nodes with 16MB each. The size may be specified in bytes (without any unit suffix) or as TB, GB, MB or kB.
- `PU:2(indexes=0,2,1,3)` specifies 2 PU children and the full list of OS indexes among the entire set of 4 PU objects.
- `PU:2(indexes=numa:core)` specifies 2 PU children whose OS indexes are interleaved by NUMA node first and then by package.
- Attributes in parentheses at the very beginning of the description apply to the root object.

## 11.2 Loading a synthetic topology

Aside from lstopo, the hwloc programming interface offers the same ability by passing the synthetic description string to hwloc_topology_set_synthetic() before hwloc_topology_load().

Synthetic topologies are created by the `synthetic` component. This component may be enabled by force by setting the HWLOC_SYNTHETIC environment variable to something such as `node:2 core:3 pu:4`.

Loading a synthetic topology disables binding support since the topology usually does not match the underlying hardware. Binding may be reenabled as usual by setting HWLOC_THISSYSTEM=1 in the environment or by setting the HWLOC_TOPOLOGY_FLAG_IS_THISSYSTEM topology flag.

## 11.3 Exporting a topology as a synthetic string

The function hwloc_topology_export_synthetic() may export a topology as a synthetic string. It offers a convenient way to quickly describe the contents of a machine. The lstopo tool may also perform such an export by forcing the output format.

```
$ lstopo --of synthetic --no-io
Package:1 L3Cache:1 L2Cache:2 L1dCache:1 L1iCache:1 Core:1 PU:2
```

The exported string may be passed back to hwloc for recreating another similar topology (see also Are synthetic strings compatible between hwloc releases?). The entire tree will be similar, but some attributes such as the processor model will be missing.

Such an export is only possible if the topology is totally symmetric. It means that the `symmetric_subtree` field of the root object is set. Also memory children should be attached in a symmetric way (e.g. the same number of memory children below each Package object, etc.). However, I/O devices and Misc objects are ignored when looking at symmetry and exporting the string.

# Chapter 12

# Interoperability With Other Software

Although hwloc offers its own portable interface, it still may have to interoperate with specific or non-portable libraries that manipulate similar kinds of objects. hwloc therefore offers several specific "helpers" to assist converting between those specific interfaces and hwloc.

Some external libraries may be specific to a particular OS; others may not always be available. The hwloc core therefore generally does not explicitly depend on these types of libraries. However, when a custom application uses or otherwise depends on such a library, it may optionally include the corresponding hwloc helper to extend the hwloc interface with dedicated helpers.

Most of these helpers use structures that are specific to these external libraries and only meaningful on the local machine. If so, the helper requires the input topology to match the current machine. Some helpers also require I/O device discovery to be supported and enabled for the current topology.

**Linux specific features** hwloc/linux.h offers Linux-specific helpers that utilize some non-portable features of the Linux system, such as binding threads through their thread ID ("tid") or parsing kernel CPU mask files.

**Linux libnuma** hwloc/linux-libnuma.h provides conversion helpers between hwloc CPU sets and libnuma-specific types, such as bitmasks. It helps you use libnuma memory-binding functions with hwloc CPU sets.

**Glibc** hwloc/glibc-sched.h offers conversion routines between Glibc and hwloc CPU sets in order to use hwloc with functions such as sched_getaffinity() or pthread_attr_setaffinity_np().

**OpenFabrics Verbs** hwloc/openfabrics-verbs.h helps interoperability with the OpenFabrics Verbs interface. For example, it can return a list of processors near an OpenFabrics device. It may also return the corresponding OS device hwloc object for further information (if I/O device discovery is enabled).

**OpenCL** hwloc/opencl.h enables interoperability with the OpenCL interface. Only the AMD and NVIDIA implementations currently offer locality information. It may return the list of processors near a GPU given as a `cl_device_id`. It may also return the corresponding OS device hwloc object for further information (if I/O device discovery is enabled).

**NVIDIA CUDA** hwloc/cuda.h and hwloc/cudart.h enable interoperability with NVIDIA CUDA Driver and Runtime interfaces. For instance, it may return the list of processors near NVIDIA GPUs. It may also return the corresponding OS device hwloc object for further information (if I/O device discovery is enabled).

**NVIDIA Management Library (NVML)** hwloc/nvml.h enables interoperability with the NVIDIA NVML interface. It may return the list of processors near a NVIDIA GPU given as a `nvmlDevice_t`. It may also return the corresponding OS device hwloc object for further information (if I/O device discovery is enabled).

**NVIDIA displays** hwloc/gl.h enables interoperability with NVIDIA displays using the NV-CONTROL X extension (NVCtrl library). If I/O device discovery is enabled, it may return the OS device hwloc object that corresponds to a display given as a name such as *:0.0* or given as a port/device pair (server/screen).

**Taskset command-line tool** The taskset command-line tool is widely used for binding processes. It manipulates CPU set strings in a format that is slightly different from hwloc's one (it does not divide the string in fixed-size subsets and separates them with commas). To ease interoperability, hwloc offers routines to convert hwloc CPU sets from/to taskset-specific string format. Most hwloc command-line tools also support the `\--taskset` option to manipulate taskset-specific strings.

# Chapter 13

# Thread Safety

Like most libraries that mainly fill data structures, hwloc is not thread safe but rather reentrant: all state is held in a hwloc_topology_t instance without mutex protection. That means, for example, that two threads can safely operate on and modify two different hwloc_topology_t instances, but they should not simultaneously invoke functions that modify the *same* instance. Similarly, one thread should not modify a hwloc_topology_t instance while another thread is reading or traversing it. However, two threads can safely read or traverse the same hwloc_topology_t instance concurrently.

When running in multiprocessor environments, be aware that proper thread synchronization and/or memory coherency protection is needed to pass hwloc data (such as hwloc_topology_t pointers) from one processor to another (e.g., a mutex, semaphore, or a memory barrier). Note that this is not a hwloc-specific requirement, but it is worth mentioning.

For reference, hwloc_topology_t modification operations include (but may not be limited to):

**Creation and destruction** `hwloc_topology_init()`, `hwloc_topology_load()`, `hwloc_topology_destroy()` (see Topology Creation and Destruction) imply major modifications of the structure, including freeing some objects. No other thread cannot access the topology or any of its objects at the same time.

Also references to objects inside the topology are not valid anymore after these functions return.

**Runtime topology modifications** `hwloc_topology_insert_misc_object()`, `hwloc_topology_alloc_group_object()`, and `hwloc_topology_insert_group_object()` (see Modifying a loaded Topology) may modify the topology significantly by adding objects inside the tree, changing the topology depth, etc.

`hwloc_distances_add()` and `hwloc_distances_remove()` (see Add or remove distances between objects) modify the list of distance structures in the topology, and the former may even insert new Group objects.

`hwloc_topology_restrict()` modifies the topology even more dramatically by removing some objects.

Although references to former objects *may* still be valid after insertion or restriction, it is strongly advised to not rely on any such guarantee and always re-consult the topology to reacquire new instances of objects.

**Consulting distances** `hwloc_distances_get()` and its variants are thread-safe except if the topology was recently modified (because distances may involve objects that were removed).

Whenever the topology is modified (see above), one dummy (but valid) `hwloc_distances_get()` call should be performed in the same thread-safe context to force the refresh of internal distances structures.

Once this refresh has been performed, multiple `hwloc_distances_get()` may then be performed concurrently by multiple threads.

**Locating topologies** `hwloc_topology_set_*` (see Topology Detection Configuration and Query) do not modify the topology directly, but they do modify internal structures describing the behavior of the upcoming invocation of `hwloc_topology_load()`. Hence, all of these functions should not be used concurrently.

# Chapter 14

# Components and plugins

hwloc is organized in components that are responsible for discovering objects. Depending on the topology configuration, some components will be used, some will be ignored. The usual default is to enable the native operating system component, (e.g. `linux` or `solaris`) and the `pci` miscellaneous component. If available, an architecture-specific component (such as `x86`) may also improve the topology detection.

If a XML topology is loaded, the `xml` discovery component will be used instead of all other components. It internally uses a specific class of components for the actual XML import/export routines (`xml_libxml` and `xml_nolibxml`) but these will not be discussed here (see libxml2 and minimalistic XML backends).

## 14.1 Components enabled by default

The hwloc core contains a list of components sorted by priority. Each one is enabled as long as it does not conflict with the previously enabled ones. This includes native operating system components, architecture-specific ones, and if available, I/O components such as `pci`.

Usually the native operating system component (when it exists, e.g. `linux` or `aix`) is enabled first. Then hwloc looks for an architecture specific component (e.g. `x86`). Finally there also exist a basic component (`no_os`) that just tries to discover the number of PUs in the system.

Each component discovers as much topology information as possible. Most of them, including most native OS components, do nothing unless the topology is still empty. Some others, such as `x86` and `pci`, can complete and annotate what other backends found earlier. Discovery is performed by phases: CPUs are first discovered, then memory is attached, then PCI, etc.

Default priorities ensure that clever components are invoked first. Native operating system components have higher priorities, and are therefore invoked first, because they likely offer very detailed topology information. If needed, it will be later extended by architecture-specific information (e.g. from the `x86` component).

If any configuration function such as hwloc_topology_set_xml() is used before loading the topology, the corresponding component is enabled first. Then, as usual, hwloc enables any other component (based on priorities) that does not conflict.

Certain components that manage a virtual topology, for instance XML topology import or synthetic topology description, conflict with all other components. Therefore, one of them may only be loaded (e.g. with `hwloc_topology_set_xml()`) if no other component is enabled.

The environment variable `HWLOC_COMPONENTS_VERBOSE` may be set to get verbose messages about component registration (including their priority) and enabling.

## 14.2 Selecting which components to use

If no topology configuration functions such as `hwloc_topology_set_synthetic()` have been called, plugins may be selected with environment variables such as `HWLOC_XMLFILE`, `HWLOC_SYNTHETIC`, `HWLOC_FSROOT`, or `HWLOC_CPUID_PATH` (see Environment Variables).

Finally, the environment variable `HWLOC_COMPONENTS` resets the list of selected components. If the variable is set and empty (or set to a single comma separating nothing, since some operating systems do not accept empty variables), the normal plugin priority order is used.

If the variable is set to `x86` in this variable will cause the `x86` component to take precedence over any other component, including the native operating system component. It is therefore loaded first, before hwloc tries to load all remaining non-conflicting components. In this case, `x86` would take care of discovering everything it supports, instead of only completing what the native OS information. This may be useful if the native component is buggy on some platforms.

It is possible to prevent some components from being loaded by prefixing their name with - in the list. For instance `x86,-pci` will load the `x86` component, then let hwloc load all the usual components except `pci`. A single component phase may also be blacklisted, for instance with `-linux:io`. hwloc_topology_set_components() may also be used inside the program to prevent the loading of a specific component (or phases) for the target topology.

It is possible to prevent all remaining components from being loaded by placing `stop` in the environment variable. Only the components listed before this keyword will be enabled.

## 14.3 Loading components from plugins

Components may optionally be built as plugins so that the hwloc core library does not directly depend on their dependencies (for instance the `libpciaccess` library). Plugin support may be enabled with the `\--enable-plugins` configure option. All components buildable as plugins will then be built as plugins. The configure option may be given a comma-separated list of component names to specify the exact list of components to build as plugins.

Plugins are built as independent dynamic libraries that are installed in `$libdir/hwloc`. All plugins found in this directory are loaded during `topology_init()` (unless blacklisted in `HWLOC_PLUGINS_BLACKLIST`, see Environment Variables). A specific list of directories (colon-separated) to scan may be specified in the `HWLOC_PLUGINS_PATH` environment variable.

Note that loading a plugin just means that the corresponding component is registered to the hwloc core. Components are then only enabled if the topology configuration requests it, as explained in the previous sections.

Also note that plugins should carefully be enabled and used when embedding hwloc in another project, see Embedding hwloc in Other Software for details.

## 14.4 Existing components and plugins

All components distributed within hwloc are listed below. The list of actually available components may be listed at running with the `HWLOC_COMPONENTS_VERBOSE` environment variable (see Environment Variables).

**linux** The official component for discovering CPU, memory and I/O devices on Linux. It discovers PCI devices without the help of external libraries such as libpciaccess, but requires the pci component for adding vendor/device names to PCI objects. It also discovers many kinds of Linux-specific OS devices.

**aix, darwin, freebsd, hpux, netbsd, solaris, windows** Each officially supported operating system has its own native component, which is statically built when supported, and which is used by default.

**x86** The x86 architecture (either 32 or 64 bits) has its own component that may complete or replace the previously-found CPU information. It is statically built when supported.

**bgq** This component is specific to IBM BlueGene/Q compute node (running CNK). It is built and enabled by default when `\--host=powerpc64-bgq-linux` is passed to configure (see How do I build hwloc for BlueGene/Q?).

**no_os** A basic component that just tries to detect the number of processing units in the system. It mostly serves on operating systems that are not natively supported. It is always statically built.

**pci** PCI object discovery uses the external pciaccess library (aka libpciaccess); see I/O Devices. It may also annotate existing PCI devices with vendor and device names. **It may be built as a plugin**.

**opencl** The OpenCL component creates co-processor OS device objects such as *opencl0d0* (first device of the first OpenCL platform) or *opencl1d3* (fourth device of the second platform). Only the AMD OpenCL implementation currently offers locality information. **It may be built as a plugin**.

**cuda** This component creates co-processor OS device objects such as *cuda0* that correspond to NVIDIA GPUs used with CUDA library. **It may be built as a plugin**.

**nvml** Probing the NVIDIA Management Library creates OS device objects such as *nvml0* that are useful for batch schedulers. It also detects the actual PCIe link bandwidth without depending on power management state and without requiring administrator privileges. **It may be built as a plugin**.

**gl** Probing the NV-CONTROL X extension (NVCtrl library) creates OS device objects such as *:0.0* corresponding to NVIDIA displays. They are useful for graphical applications that need to place computation and/or data near a rendering GPU. **It may be built as a plugin**.

**synthetic** Synthetic topology support (see Synthetic topologies) is always built statically.

**xml** XML topology import (see Importing and exporting topologies from/to XML files) is always built statically. It internally uses one of the XML backends (see libxml2 and minimalistic XML backends).

- **xml_nolibxml** is a basic and hwloc-specific XML import/export. It is always statically built.
- **xml_libxml** relies on the external libxml2 library for provinding a feature-complete XML import/export. **It may be built as a plugin**.

**fake** A dummy plugin that does nothing but is used for debugging plugin support.

# Chapter 15

# Embedding hwloc in Other Software

It can be desirable to include hwloc in a larger software package (be sure to check out the LICENSE file) so that users don't have to separately download and install it before installing your software. This can be advantageous to ensure that your software uses a known-tested/good version of hwloc, or for use on systems that do not have hwloc pre-installed.

When used in "embedded" mode, hwloc will:

- not install any header files
- not build any documentation files
- not build or install any executables or tests
- not build `libhwloc.*` -- instead, it will build `libhwloc_embedded.*`

There are two ways to put hwloc into "embedded" mode. The first is directly from the configure command line:

```
shell$ ./configure --enable-embedded-mode ...
```

The second requires that your software project uses the GNU Autoconf / Automake / Libtool tool chain to build your software. If you do this, you can directly integrate hwloc's m4 configure macro into your configure script. You can then invoke hwloc's configuration tests and build setup by calling an m4 macro (see below).

Although hwloc dynamic shared object plugins may be used in embedded mode, the embedder project will have to manually setup dlopen or libltdl in its build system so that hwloc can load its plugins at run time. Also, embedders should be aware of complications that can arise due to public and private linker namespaces (e.g., if the embedder project is loaded into a private namespace and then hwloc tries to dynamically load its plugins, such loading may fail since the hwloc plugins can't find the hwloc symbols they need). The embedder project is **strongly** advised not to use hwloc's dynamically loading plugins / dlopen / libltdl capability.

## 15.1 Using hwloc's M4 Embedding Capabilities

Every project is different, and there are many different ways of integrating hwloc into yours. What follows is *one* example of how to do it.

If your project uses recent versions Autoconf, Automake, and Libtool to build, you can use hwloc's embedded m4 capabilities. We have tested the embedded m4 with projects that use Autoconf 2.65, Automake 1.11.1, and Libtool 2.2.6b. Slightly earlier versions of may also work but are untested. Autoconf versions prior to 2.65 are almost certain to not work.

You can either copy all the config/hwloc∗m4 files from the hwloc source tree to the directory where your project's m4 files reside, or you can tell aclocal to find more m4 files in the embedded hwloc's "config" subdirectory (e.g., add "-Ipath/to/embedded/hwloc/config" to your Makefile.am's ACLOCAL_AMFLAGS).

The following macros can then be used from your configure script (only HWLOC_SETUP_CORE *must* be invoked if using the m4 macros):

- HWLOC_SETUP_CORE(config-dir-prefix, action-upon-success, action-upon-failure, print_-banner_or_not): Invoke the hwloc configuration tests and setup the hwloc tree to build. The first argument is the prefix to use for AC_OUTPUT files -- it's where the hwloc tree is located relative to `$top_srcdir`. Hence, if your embedded hwloc is located in the source tree at contrib/hwloc, you should pass `[contrib/hwloc]` as the first argument. If HWLOC_SETUP_CORE and the

rest of `configure` completes successfully, then "make" traversals of the hwloc tree with standard Automake targets (all, clean, install, etc.) should behave as expected. For example, it is safe to list the hwloc directory in the SUBDIRS of a higher-level Makefile.am. The last argument, if not empty, will cause the macro to display an announcement banner that it is starting the hwloc core configuration tests.

HWLOC_SETUP_CORE will set the following environment variables and AC_SUBST them: HWLOC_EMBEDDED_CFLAGS, HWLOC_EMBEDDED_CPPFLAGS, and HWLOC_EMBEDDED_-LIBS. These flags are filled with the values discovered in the hwloc-specific m4 tests, and can be used in your build process as relevant. The _CFLAGS, _CPPFLAGS, and _LIBS variables are necessary to build libhwloc (or libhwloc_embedded) itself.

HWLOC_SETUP_CORE also sets HWLOC_EMBEDDED_LDADD environment variable (and AC_-SUBSTs it) to contain the location of the libhwloc_embedded.la convenience Libtool archive. It can be used in your build process to link an application or other library against the embedded hwloc library.

**NOTE: If the HWLOC_SET_SYMBOL_PREFIX macro is used, it must be invoked *before* HWLOC_SETUP_CORE.**

- HWLOC_BUILD_STANDALONE: HWLOC_SETUP_CORE defaults to building hwloc in an "embedded" mode (described above). If HWLOC_BUILD_STANDALONE is invoked ∗before∗ HWLOC_SETUP_CORE, the embedded definitions will not apply (e.g., libhwloc.la will be built, not libhwloc_embedded.la).

- HWLOC_SET_SYMBOL_PREFIX(foo_): Tells the hwloc to prefix all of hwloc's types and public symbols with "foo_"; meaning that function hwloc_init() becomes foo_hwloc_init(). Enum values are prefixed with an upper-case translation if the prefix supplied; HWLOC_OBJ_CORE becomes FOO_hwloc_OBJ_CORE. This is recommended behavior if you are including hwloc in middleware -- it is possible that your software will be combined with other software that links to another copy of hwloc. If both uses of hwloc utilize different symbol prefixes, there will be no type/symbol clashes, and everything will compile, link, and run successfully. If you both embed hwloc without changing the symbol prefix and also link against an external hwloc, you may get multiple symbol definitions when linking your final library or application.

- HWLOC_SETUP_DOCS, HWLOC_SETUP_UTILS, HWLOC_SETUP_TESTS: These three macros only apply when hwloc is built in "standalone" mode (i.e., they should NOT be invoked unless HWLOC_BUILD_STANDALONE has already been invoked).

- HWLOC_DO_AM_CONDITIONALS: If you embed hwloc in a larger project and build it conditionally with Automake (e.g., if HWLOC_SETUP_CORE is invoked conditionally), you must unconditionally invoke HWLOC_DO_AM_CONDITIONALS to avoid warnings from Automake (for the cases where hwloc is not selected to be built). This macro is necessary because hwloc uses some AM_CONDITIONALs to build itself, and AM_CONDITIONALs cannot be defined conditionally. Note that it is safe (but unnecessary) to call HWLOC_DO_AM_CONDITIONALS even if HWLOC_SETUP_CORE is invoked unconditionally. If you are not using Automake to build hwloc, this macro is unnecessary (and will actually cause errors because it invoked AM_∗ macros that will be undefined).

**NOTE:** When using the HWLOC_SETUP_CORE m4 macro, it may be necessary to explicitly invoke AC_CANONICAL_TARGET (which requires config.sub and config.guess) and/or AC_USE_-SYSTEM_EXTENSIONS macros early in the configure script (e.g., after AC_INIT but before AM_INIT_-AUTOMAKE). See the Autoconf documentation for further information.

Also note that hwloc's top-level configure.ac script uses exactly the macros described above to build hwloc in a standalone mode (by default). You may want to examine it for one example of how these macros are used.

## 15.2 Example Embedding hwloc

Here's an example of integrating with a larger project named sandbox that already uses Autoconf, Automake, and Libtool to build itself:

```
# First, cd into the sandbox project source tree
shell$ cd sandbox
shell$ cp -r /somewhere/else/hwloc-<version> my-embedded-hwloc
shell$ edit Makefile.am
  1. Add "-Imy-embedded-hwloc/config" to ACLOCAL_AMFLAGS
  2. Add "my-embedded-hwloc" to SUBDIRS
  3. Add "$(HWLOC_EMBEDDED_LDADD)" and "$(HWLOC_EMBEDDED_LIBS)" to
     sandbox's executable's LDADD line.  The former is the name of the
     Libtool convenience library that hwloc will generate.  The latter
     is any dependent support libraries that may be needed by
     $(HWLOC_EMBEDDED_LDADD).
  4. Add "$(HWLOC_EMBEDDED_CFLAGS)" to AM_CFLAGS
  5. Add "$(HWLOC_EMBEDDED_CPPFLAGS)" to AM_CPPFLAGS
shell$ edit configure.ac
  1. Add "HWLOC_SET_SYMBOL_PREFIX(sandbox_hwloc_)" line
  2. Add "HWLOC_SETUP_CORE([my-embedded-hwloc], [happy=yes], [happy=no])" line
  3. Add error checking for happy=no case
shell$ edit sandbox.c
  1. Add #include <hwloc.h>
  2. Add calls to sandbox_hwloc_init() and other hwloc API functions
```

Now you can bootstrap, configure, build, and run the sandbox as normal -- all calls to "sandbox_hwloc_*" will use the embedded hwloc rather than any system-provided copy of hwloc.

# Chapter 16

# Frequently Asked Questions

## 16.1 Concepts

### 16.1.1 I only need binding, why should I use hwloc ?

hwloc is its portable API that works on a variety of operating systems. It supports binding of threads, processes and memory buffers (see CPU binding and Memory binding). Even if some features are not supported on some systems, using hwloc is much easier than reimplementing your own portability layer.

Moreover, hwloc provides knowledge of cores and hardware threads. It offers easy ways to bind tasks to individual hardware threads, or to entire multithreaded cores, etc. See How may I ignore symmetric multithreading, hyper-threading, etc. in hwloc?. Most alternative software for binding do not even know whether each core is single-threaded, multithreaded or hyper-threaded. They would bind to individual threads without any way to know whether multiple tasks are in the same physical core.

However, using hwloc comes with an overhead since a topology must be loaded before gathering information and binding tasks or memory. Fortunately this overhead may be significantly reduced by filtering non-interesting information out of the topology. For instance the following code builds a topology that only contains Cores (explicitly filtered-in below), hardware threads (PUs, cannot be filtered-out), NUMA nodes (cannot be filtered-out), and the root object (usually a Machine; the root cannot be removed without breaking the tree).

```
hwloc_topology_t topology;
hwloc_topology_init(&topology);
/* filter everything out */
hwloc_topology_set_all_types_filter(topology, HWLOC_TYPE_FILTER_KEEP_NONE);
/* filter Cores back in */
hwloc_topology_set_type_filter(topology, HWLOC_OBJ_CORE, HWLOC_TYPE_FILTER_KEEP_ALL);
hwloc_topology_load(topology);
```

However, one should remember that filtering such objects out removes locality information from the hwloc tree. For instance, we do not know anymore which PU is close to which NUMA node. This would be useful to applications that explicitly want to place specific memory buffers close to specific tasks. Those applications just need to tell hwloc to keep Group objects that bring structure information:

```
hwloc_topology_set_type_filter(topology, HWLOC_OBJ_GROUP, HWLOC_TYPE_FILTER_KEEP_STRUCTURE);
```

Note that the default configuration is to keep all objects enabled, except I/Os and instruction caches.

### 16.1.2 Should I use logical or physical/OS indexes? and how?

One of the original reasons why hwloc was created is that **physical/OS indexes** (`obj->os_index`) are often crazy and unpredictable: logical processors numbers are usually non-contiguous (processors 0 and 1 are not physically close), they vary from one machine to another, and may even change after a BIOS or system update. This numbers make task placement hardly portable. Moreover some objects have no physical/OS numbers (caches), and some objects have non-unique numbers (core numbers are only unique within a socket). Physical/OS indexes are only guaranteed to exist and be unique for PU and NUMA nodes.

hwloc therefore introduces **logical indexes** (`obj->logical_index`) which are portable, contiguous and logically ordered (based on the resource organization in the locality tree). In general, one should only use logical indexes and just let hwloc do the internal conversion when really needed (when talking to the OS and hardware).

hwloc developers recommends that users do not use physical/OS indexes unless they really know what they are doing. The main reason for still using physical/OS indexes is when interacting with non-hwloc tools such as numactl or taskset, or when reading hardware information from raw sources such as /proc/cpuinfo.

lstopo options -l and -p may be used to switch between logical indexes (prefixed with L#) and physical/OS indexes (P#). Converting one into the other may also be achieved with hwloc-calc which may manipulate either logical or physical indexes as input or output. See also hwloc-calc.

```
# Convert PU with physical number 3 into logical number
$ hwloc-calc -I pu --physical-input --logical-output pu:3
5

# Convert a set of NUMA nodes from logical to physical
# (beware that the output order may not match the input order)
$ hwloc-calc -I numa --logical-input --physical-output numa:2-3 numa:7
0,2,5
```

### 16.1.3 hwloc is only a structural model, it ignores performance models, memory bandwidth, etc.?

hwloc is indeed designed to provide applications with a structural model of the platform. This is an orthogonal approach to describing the machine with performance models, for instance using memory bandwidth or latencies measured by benchmarks. We believe that both approaches are important for helping application make the most of the hardware.

For instance, on a dual-processor host with four cores each, hwloc clearly shows which four cores are together. Latencies between all pairs of cores of the same processor are likely identical, and also likely lower than the latency between cores of different processors. However, the structural model cannot guarantee such implementation details. On the other side, performance models would reveal such details without always clearly identifying which cores are in the same processor.

The focus of hwloc is mainly of the structural modeling side. However, hwloc lets user adds performance information to the topology through distances (see Retrieve distances between objects and Add or remove distances between objects) or even custom annotations (see How do I annotate the topology with private notes?). hwloc may also use such distance information for grouping objects together (see hwloc only has a one-dimensional view of the architecture, it ignores distances? and What are these Group objects in my topology?).

### 16.1.4 hwloc only has a one-dimensional view of the architecture, it ignores distances?

hwloc places all objects in a tree. Each level is a one-dimensional view of a set of similar objects. All children of the same object (siblings) are assumed to be equally interconnected (same distance between any of them), while the distance between children of different objects (cousins) is supposed to be larger.

Modern machines exhibit complex hardware interconnects, so this tree may miss some information about the actual physical distances between objects. The hwloc topology may therefore be annotated with distance information that may be used to build a more realistic representation (multi-dimensional) of each level. For instance, there can be a distance matrix that representing the latencies between any pair of NUMA nodes if the BIOS and/or operating system reports them.

For more information about the distance API, see Retrieve distances between objects and Add or remove distances between objects.

### 16.1.5 What are these Group objects in my topology?

hwloc comes with a set of predefined object types (Core, Package, NUMA node, Caches) that match the vast majority of hardware platforms. The HWLOC_OBJ_GROUP type was designed for cases where this

set is not sufficient. Groups may be used anywhere to add more structure information to the topology, for instance to show that 2 out of 4 NUMA nodes are actually closer than the others. When applicable, the `subtype` field describes why a Group was actually added (see also Normal attributes).

hwloc currently uses Groups for the following reasons:

- NUMA parents when memory locality does not match any existing object.
- I/O parents when I/O locality does not match any existing object.
- Distance-based groups made of close objects.
- AMD Bulldozer dual-core compute units (`subtype` is `ComputeUnit`, in the x86 backend), but these objects are usually merged with the L2 caches.
- Intel Extended Topology Enumeration levels (in the x86 backend).
- Windows processor groups (unless they contain a single NUMA node, or a single Package, etc.).
- IBM S/390 "Books" on Linux (`subtype` is `Book`).
- AIX unknown hierarchy levels.

hwloc Groups are only kept if no other object has the same locality information. It means that a Group containing a single child is merged into that child. And a Group is merged into its parent if it is its only child. For instance a Windows processor group containing a single NUMA node would be merged with that NUMA node since it already contains the relevant hierarchy information.

When inserting a custom Group with hwloc_hwloc_topology_insert_group_object(), this merging may be disabled by setting its `dont_merge` attribute.

### 16.1.6 What happens if my topology is asymmetric?

hwloc supports asymmetric topologies even if most platforms are usually symmetric. For example, there could be different types of processors in a single machine, each with different numbers of cores, symmetric multithreading, or levels of caches.

In practice, asymmetric topologies mostly appear when intermediate groups are added for I/O affinity: on a 4-package machine, an I/O bus may be connected to 2 packages. These packages are below an additional Group object, while the other packages are not (see also What are these Group objects in my topology?).

To understand how hwloc manages such cases, one should first remember the meaning of levels and cousin objects. All objects of the same type are gathered as horizontal levels with a given depth. They are also connected through the cousin pointers of the hwloc_obj structure. Object attribute (cache depth and type, group depth) are also taken in account when gathering objects as horizontal levels. To be clear: there will be one level for L1i caches, another level for L1d caches, another one for L2, etc.

If the topology is asymmetric (e.g., if a group is missing above some processors), a given horizontal level will still exist if there exist any objects of that type. However, some branches of the overall tree may not have an object located in that horizontal level. Note that this specific hole within one horizontal level does not imply anything for other levels. All objects of the same type are gathered in horizontal levels even if their parents or children have different depths and types.

See the diagram in Terms and Definitions for a graphical representation of such topologies.

Moreover, it is important to understand that a same parent object may have children of different types (and therefore, different depths). **These children are therefore siblings (because they have the same parent), but they are *not* cousins (because they do not belong to the same horizontal level).**

### 16.1.7 What happens to my topology if I disable symmetric multithreading, hyper-threading, etc. in the system?

hwloc creates one PU (processing unit) object per hardware thread. If your machine supports symmetric multithreading, for instance Hyper-Threading, each Core object may contain multiple PU objects:

```
$ lstopo -
...
  Core L#0
    PU L#0 (P#0)
    PU L#1 (P#2)
  Core L#1
    PU L#2 (P#1)
    PU L#3 (P#3)
```

x86 machines usually offer the ability to disable hyper-threading in the BIOS. Or it can be disabled on the Linux kernel command-line at boot time, or later by writing in sysfs virtual files.

If you do so, the hwloc topology structure does not significantly change, but some PU objects will not appear anymore. No level will disappear, you will see the same number of Core objects, but each of them will contain a single PU now. The PU level does not disappear either (remember that hwloc topologies always contain a PU level at the bottom of the topology) even if there is a single PU object per Core parent.

```
$ lstopo -
...
  Core L#0
    PU L#0 (P#0)
  Core L#1
    PU L#1 (P#1)
```

### 16.1.8 How may I ignore symmetric multithreading, hyper-threading, etc. in hwloc?

First, see What happens to my topology if I disable symmetric multithreading, hyper-threading, etc. in the system? for more information about multithreading.

If you need to ignore symmetric multithreading in software, you should likely manipulate hwloc Core objects directly:

```
/* get the number of cores */
unsigned nbcores = hwloc_get_nbobjs_by_type(topology, HWLOC_OBJ_CORE);
...
/* get the third core below the first package */
hwloc_obj_t package, core;
package = hwloc_get_obj_by_type(topology, HWLOC_OBJ_PACKAGE, 0);
core = hwloc_get_obj_inside_cpuset_by_type(topology, package->cpuset,
                                           HWLOC_OBJ_CORE, 2);
```

Whenever you want to bind a process or thread to a core, make sure you singlify its cpuset first, so that the task is actually bound to a single thread within this core (to avoid useless migrations).

```
/* bind on the second core */
hwloc_obj_t core = hwloc_get_obj_by_type(topology, HWLOC_OBJ_CORE, 1);
hwloc_cpuset_t set = hwloc_bitmap_dup(core->cpuset);
hwloc_bitmap_singlify(set);
hwloc_set_cpubind(topology, set, 0);
hwloc_bitmap_free(set);
```

With hwloc-calc or hwloc-bind command-line tools, you may specify that you only want a single-thread within each core by asking for their first PU object:

```
$ hwloc-calc core:4-7
0x0000ff00
$ hwloc-calc core:4-7.pu:0
0x00005500
```

When binding a process on the command-line, you may either specify the exact thread that you want to use, or ask hwloc-bind to singlify the cpuset before binding

```
$ hwloc-bind core:3.pu:0 -- echo "hello from first thread on core #3"
hello from first thread on core #3
...
$ hwloc-bind core:3 --single -- echo "hello from a single thread on core #3"
hello from a single thread on core #3
```

## 16.2 Advanced

### 16.2.1 I do not want hwloc to rediscover my enormous machine topology every time I rerun a process

Although the topology discovery is not expensive on common machines, its overhead may become significant when multiple processes repeat the discovery on large machines (for instance when starting one process per core in a parallel application). The machine topology usually does not vary much, except if some cores are stopped/restarted or if the administrator restrictions are modified. Thus rediscovering the whole topology again and again may look useless.

For this purpose, hwloc offers XML import/export and shared memory features.

XML lets you save the discovered topology to a file (for instance with the lstopo program) and reload it later by setting the HWLOC_XMLFILE environment variable. The HWLOC_THISSYSTEM environment variable should also be set to 1 to assert that loaded file is really the underlying system.

Loading a XML topology is usually much faster than querying multiple files or calling multiple functions of the operating system. It is also possible to manipulate such XML files with the C programming interface, and the import/export may also be directed to memory buffer (that may for instance be transmitted between applications through a package). See also Importing and exporting topologies from/to XML files.

**Note:**

> The environment variable HWLOC_THISSYSTEM_ALLOWED_RESOURCES may be used to load a XML topology that contains the entire machine and restrict it to the part that is actually available to the current process (e.g. when Linux Cgroup/Cpuset are used to restrict the set of resources). See Environment Variables.

Shared-memory topologies consist in one process exposing its topology in a shared-memory buffer so that other processes (running on the same machine) may use it directly. This has the advantage of reducing the memory footprint since a single topology is stored in physical memory for multiple processes. However, it requires all processes to map this shared-memory buffer at the same virtual address, which may be difficult in some cases. This API is described in Sharing topologies between processes.

### 16.2.2 How many topologies may I use in my program?

hwloc lets you manipulate multiple topologies at the same time. However, these topologies consume memory and system resources (for instance file descriptors) until they are destroyed. It is therefore discouraged to open the same topology multiple times.

Sharing a single topology between threads is easy (see Thread Safety) since the vast majority of accesses are read-only.

If multiple topologies of different (but similar) nodes are needed in your program, have a look at How to avoid memory waste when manipulating multiple similar topologies?.

### 16.2.3 How to avoid memory waste when manipulating multiple similar topologies?

hwloc does not share information between topologies. If multiple similar topologies are loaded in memory, for instance the topologies of different identical nodes of a cluster, lots of information will be duplicated.

hwloc/diff.h (see also Topology differences) offers the ability to compute topology differences, apply or unapply them, or export/import to/from XML. However, this feature is limited to basic differences such as attribute changes. It does not support complex modifications such as adding or removing some objects.

### 16.2.4 How do I annotate the topology with private notes?

Each hwloc object contains a `userdata` field that may be used by applications to store private pointers. This field is only valid during the lifetime of these container object and topology. It becomes invalid as soon the topology is destroyed, or as soon as the object disappears, for instance when restricting the topology. The userdata field is not exported/imported to/from XML by default since hwloc does not know what it contains. This behavior may be changed by specifying application-specific callbacks with `hwloc_topology_set_userdata_export_callback()` and `hwloc_topology_set_userdata_import_callback()`.

Each object may also contain some *info* attributes (key name and value) that are setup by hwloc during discovery and that may be extended by the user with `hwloc_obj_add_info()` (see also Object attributes). Contrary to the `userdata` field which is unique, multiple info attributes may exist for each object, even with the same name. These attributes are always exported to XML. However, only character strings may be used as key names and values.

It is also possible to insert Misc objects with a custom name anywhere as a leaf of the topology (see Miscellaneous objects). And Misc objects may have their own userdata and info attributes just like any other object.

The hwloc-annotate command-line tool may be used for adding Misc objects and info attributes.

There is also a topology-specific userdata pointer that can be used to recognize different topologies by storing a custom pointer. It may be manipulated with `hwloc_topology_set_userdata()` and `hwloc_topology_get_userdata()`.

## 16.3 Caveats

### 16.3.1 Why is hwloc slow?

Building a hwloc topology on a large machine may be slow because the discovery of hundreds of hardware cores or threads takes time (especially when reading thousands of sysfs files on Linux). Ignoring some objects (for instance caches) that aren't useful to the current application may improve this overhead (see I only need binding, why should I use hwloc ?). One should also consider using XML (see I do not want hwloc to rediscover my enormous machine topology every time I rerun a process) to work around such issues.

Additionally, lstopo enables most hwloc objects and discovery flags by default so that the output topology is as precise as possible (while hwloc disables many of them by default). This includes I/O device discovery through PCI libraries as well as external libraries such as NVML. To speed up lstopo, you may disable such features with command-line options such as `\--no-io`.

When NVIDIA GPU probing is enabled with CUDA or NVML, one should make sure that the *Persistent* mode is enabled (with `nvidia-smi -pm 1`) to avoid significant GPU initialization overhead.

When AMD GPU discovery is enabled with OpenCL and hwloc is used remotely over ssh, some spurious round-trips on the network may significantly increase the discovery time. Forcing the `DISPLAY` environment variable to the remote X server display (usually `:0`) instead of only setting the `COMPUTE` variable may avoid this.

Also remember that these components may be disabled at build-time with configure flags such as `\--disable-opencl`, `\--disable-cuda` or `\--disable-nvml`, and at runtime with the environment variable `HWLOC_COMPONENTS=-opencl,-cuda,-nvml` or with hwloc_topology_set_components().

### 16.3.2 Does hwloc require privileged access?

hwloc discovers the topology by querying the operating system. Some minor features may require privileged access to the operation system. For instance memory module discovery on Linux is reserved to root, and the entire PCI discovery on Solaris and BSDs requires access to some special files that are usually restricted to root (/dev/pci∗ or /devices/pci∗).

To workaround this limitation, it is recommended to export the topology as a XML file generated by the administrator (with the lstopo program) and make it available to all users (see Importing and exporting topologies from/to XML files). It will offer all discovery information to any application without requiring any privileged access anymore. Only the necessary hardware characteristics will be exported, no sensitive information will be disclosed through this XML export.

This XML-based model also has the advantage of speeding up the discovery because reading a XML topology is usually much faster than querying the operating system again.

The utility `hwloc-dump-hwdata` is also involved in gathering privileged information at boot time and making it available to non-privileged users (note that this may require a specific SELinux MLS policy module). However, it only applies to Intel Xeon Phi processors for now (see Why do I need hwloc-dump-hwdata for memory on Intel Xeon Phi processor?). See also `HWLOC_DUMPED_HWDATA_DIR` in Environment Variables for details about the location of dumped files.

### 16.3.3 What should I do when hwloc reports "operating system" warnings?

When the operating system reports invalid locality information (because of either software or hardware bugs), hwloc may fail to insert some objects in the topology because they cannot fit in the already built tree of resources. If so, hwloc will report a warning like the following. The object causing this error is ignored, the discovery continues but the resulting topology will miss some objects and may be asymmetric (see also What happens if my topology is asymmetric?).

```
****************************************************************************
* hwloc received invalid information from the operating system.
*
* L3 (cpuset 0x000003f0) intersects with NUMANode (P#0 cpuset 0x0000003f) without inclusion!
* Error occurred in topology.c line 940
*
* Please report this error message to the hwloc user's mailing list,
* along with the files generated by the hwloc-gather-topology script.
*
```

```
* hwloc will now ignore this invalid topology information and continue.
****************************************************************************
```

These errors are common on large AMD platforms because of BIOS and/or Linux kernel bugs causing invalid L3 cache information. In the above example, the hardware reports a L3 cache that is shared by 2 cores in the first NUMA node and 4 cores in the second NUMA node. That's wrong, it should actually be shared by all 6 cores in a single NUMA node. The resulting topology will miss some L3 caches.

If your application does not care about cache sharing, or if you do not plan to request cache-aware binding in your process launcher, you may likely ignore this error (and hide it by setting HWLOC_HIDE_ERRORS=1 in your environment).

Some platforms report similar warnings about conflicting Packages and NUMANodes.

On x86 hosts, passing `HWLOC_COMPONENTS=x86` in the environment may workaround some of these issues by switching to a different way to discover the topology.

Upgrading the BIOS and/or the operating system may help. Otherwise, as explained in the message, reporting this issue to the hwloc developers (by sending the tarball that is generated by the hwloc-gather-topology script on this platform) is a good way to make sure that this is a software (operating system) or hardware bug (BIOS, etc).

See also Questions and Bugs. Opening an issue on GitHub automatically displays hints on what information you should provide when reporting such bugs.

### 16.3.4 Why does Valgrind complain about hwloc memory leaks?

If you are debugging your application with Valgrind, you want to avoid memory leak reports that are caused by hwloc and not by your program.

hwloc itself is often checked with Valgrind to make sure it does not leak memory. However, some global variables in hwloc dependencies are never freed. For instance libz allocates its global state once at startup and never frees it so that it may be reused later. Some libxml2 global state is also never freed because hwloc does not know whether it can safely ask libxml2 to free it (the application may also be using libxml2 outside of hwloc).

These unfreed variables cause leak reports in Valgrind. hwloc installs a Valgrind *suppressions* file to hide them. You should pass the following command-line option to Valgrind to use it:

```
--suppressions=/path/to/hwloc-valgrind.supp
```

## 16.4 Platform-specific

### 16.4.1 How do I find the local MCDRAM NUMA node on Intel Xeon Phi processor?

Intel Xeon Phi processors introduced a new memory architecture by possibly having two distinct local memories: some normal memory (DDR) and some high-bandwidth on-package memory (MCDRAM). Processors can be configured in various clustering modes to have up to 4 *Clusters*. Moreover, each *Cluster* (quarter, half or whole processor) of the processor may have its own local parts of the DDR and of the MCDRAM. This memory and clustering configuration may be probed by looking at MemoryMode and ClusterMode attributes, see Hardware Platform Information and doc/examples/get-knl-modes.c in the source directory.

Starting with version 2.0, hwloc properly exposes this memory configuration. DDR and MCDRAM are attached as two memory children of the same parent, DDR first, and MCDRAM second if any. Depending

on the processor configuration, that parent may be a Package, a Cache, or a Group object of type `Cluster`.

Hence cores may have one or two local NUMA nodes, listed by the core nodeset. An application may allocate local memory from a core by using that nodeset. The operating system will actually allocate from the DDR when possible, or fallback to the MCDRAM.

To allocate specifically on one of these memories, one should walk up the parent pointers until finding an object with some memory children. Looking at these memory children will give the DDR first, then the MCDRAM if any. Their nodeset may then be used for allocating or binding memory buffers.

One may also traverse the list of NUMA nodes until finding some whose cpuset matches the target core or PUs. The MCDRAM NUMA nodes may be identified thanks to the `subtype` field which is set to `MCDRAM`.

Command-line tools such as `hwloc-bind` may bind memory on the MCDRAM by using the *hbm* keyword. For instance, to bind on the first MCDRAM NUMA node:

```
$ hwloc-bind --membind --hbm numa:0 -- myprogram
$ hwloc-bind --membind numa:0 -- myprogram
```

### 16.4.2 Why do I need hwloc-dump-hwdata for memory on Intel Xeon Phi processor?

Intel Xeon Phi processors may use the on-package memory (MCDRAM) as either memory or a memory-side cache (reported as a L3 cache by hwloc by default, see `HWLOC_KNL_MSCACHE_L3` in Environment Variables). There are also several clustering modes that significantly affect the memory organization (see How do I find the local MCDRAM NUMA node on Intel Xeon Phi processor? for more information about these modes). Details about these are currently only available to privileged users. Without them, hwloc relies on a heuristic for guessing the modes.

The hwloc-dump-hwdata utility may be used to dump this privileged binary information into human-readable and world-accessible files that the hwloc library will later load. The utility should usually run as root once during boot, in order to update dumped information (stored under /var/run/hwloc by default) in case the MCDRAM or clustering configuration changed between reboots.

When SELinux MLS policy is enabled, a specific hwloc policy module may be required so that all users get access to the dumped files (in /var/run/hwloc by default). One may use hwloc policy files from the SELinux Reference Policy at `https://github.com/TresysTechnology/refpolicy-contrib` (see also the documentation at `https://github.com/TresysTechnology/refpolicy/wiki/GettingStarted`).

hwloc-dump-hwdata requires `dmi-sysfs` kernel module loaded.

The utility is currently unneeded on platforms without Intel Xeon Phi processors.

See `HWLOC_DUMPED_HWDATA_DIR` in Environment Variables for details about the location of dumped files.

### 16.4.3 How do I build hwloc for BlueGene/Q?

IBM BlueGene/Q machines run a standard Linux on the login/frontend nodes and a custom CNK (*Compute Node Kernel*) on the compute nodes.

To discover the topology of a login/frontend node, hwloc should be configured as usual, without any BlueGene/Q-specific option.

However, one would likely rather discover the topology of the compute nodes where parallel jobs are actually running. If so, hwloc must be cross-compiled with the following configuration line:

```
./configure --host=powerpc64-bgq-linux --disable-shared --enable-static \
  CPPFLAGS='-I/bgsys/drivers/ppcfloor -I/bgsys/drivers/ppcfloor/spi/include/kernel/cnk/'
```

CPPFLAGS may have to be updated if your platform headers are installed in a different directory.

### 16.4.4 How do I build hwloc for Windows?

hwloc releases are available as pre-built ZIPs for Windows on both 32bits and 64bits x86 platforms. They are built using MSYS2 and MinGW on a Windows host. Such an environment allows using the Unix-like `configure`, `make` and `make install` steps without having to tweak too many variables or options. One may look at `contrib/ci.inria.fr/job-3-mingw.sh` in the hwloc repository for an example used for nightly testing.

hwloc releases also contain a basic Microsoft Visual Studio solution under `contrib/windows/`.

### 16.4.5 How to get useful topology information on NetBSD?

The NetBSD (and FreeBSD) backend uses x86-specific topology discovery (through the x86 component). This implementation requires CPU binding so as to query topology information from each individual logical processor. This means that hwloc cannot find any useful topology information unless user-level process binding is allowed by the NetBSD kernel. The `security.models.extensions.user_set_cpu_affinity` sysctl variable must be set to 1 to do so. Otherwise, only the number of logical processors will be detected.

### 16.4.6 Why does binding fail on AIX?

The AIX operating system requires specific user capabilities for attaching processes to resource sets (CAP_NUMA_ATTACH). Otherwise functions such as hwloc_set_cpubind() fail (return -1 with errno set to EPERM).

This capability must also be inherited (through the additional CAP_PROPAGATE capability) if you plan to bind a process before forking another process, for instance with `hwloc-bind`.

These capabilities may be given by the administrator with:

```
chuser "capabilities=CAP_PROPAGATE,CAP_NUMA_ATTACH" <username>
```

## 16.5 Compatibility between hwloc versions

### 16.5.1 How do I handle API changes?

The hwloc interface is extended with every new major release. Any application using the hwloc API should be prepared to check at compile-time whether some features are available in the currently installed hwloc distribution.

For instance, to check whether the hwloc version is at least 2.0, you should use:

```
#include <hwloc.h>
#if HWLOC_API_VERSION >= 0x00020000
...
#endif
```

To check for the API of release X.Y.Z at build time, you may compare HWLOC_API_VERSION with `(X<<16)+(Y<<8)+Z`.

For supporting older releases that do not have `HWLOC_OBJ_NUMANODE` and `HWLOC_OBJ_PACKAGE` yet, you may use:

```
#include <hwloc.h>
#if HWLOC_API_VERSION < 0x00010b00
#define HWLOC_OBJ_NUMANODE HWLOC_OBJ_NODE
#define HWLOC_OBJ_PACKAGE HWLOC_OBJ_SOCKET
#endif
```

Once a program is built against a hwloc library, it may also dynamically link with compatible libraries from other hwloc releases. The version of that runtime library may be queried with hwloc_get_api_version(). See How do I handle ABI breaks? for using this function for testing ABI compatibility.

### 16.5.2 What is the difference between API and library version numbers?

HWLOC_API_VERSION is the version of the API. It changes when functions are added, modified, etc. However it does not necessarily change from one release to another. For instance, two releases of the same series (e.g. 2.0.3 and 2.0.4) usually have the same HWLOC_API_VERSION (`0x00020000`). However their HWLOC_VERSION strings are different (`\"2.0.3\"` and `\"2.0.4\"` respectively).

### 16.5.3 How do I handle ABI breaks?

The hwloc interface was deeply modified in release 2.0 to fix several issues of the 1.x interface (see Upgrading to the hwloc 2.0 API and the NEWS file in the source directory for details). The ABI was broken, which means **applications must be recompiled against the new 2.0 interface**.

To check that you are not mixing old/recent headers with a recent/old runtime library, check the major revision number in the API version:

```
#include <hwloc.h>
  unsigned version = hwloc_get_api_version();
  if ((version >> 16) != (HWLOC_API_VERSION >> 16)) {
    fprintf(stderr,
            "%s compiled for hwloc API 0x%x but running on library API 0x%x.\n"
            "You may need to point LD_LIBRARY_PATH to the right hwloc library.\n"
            "Aborting since the new ABI is not backward compatible.\n",
            callname, HWLOC_API_VERSION, version);
    exit(EXIT_FAILURE);
  }
```

To specifically detect v2.0 issues:

```
#include <hwloc.h>
#if HWLOC_API_VERSION >= 0x00020000
  /* headers are recent */
  if (hwloc_get_api_version() < 0x20000)
    ... error out, the hwloc runtime library is older than 2.0 ...
#else
  /* headers are pre-2.0 */
  if (hwloc_get_api_version() >= 0x20000)
    ... error out, the hwloc runtime library is more recent than 2.0 ...
#endif
```

In theory, library sonames prevent linking with incompatible libraries. However custom hwloc installations or improperly configured build environments may still lead to such issues. Hence running one of the above (cheap) checks before initializing hwloc topology may be useful.

### 16.5.4 Are XML topology files compatible between hwloc releases?

XML topology files are forward-compatible: a XML file may be loaded by a hwloc library that is more recent than the hwloc release that exported that file.

However, hwloc XMLs are not always backward-compatible: Topologies exported by hwloc 2.x cannot be imported by 1.x by default (see XML changes for working around such issues). There are also some corner cases where backward compatibility is not guaranteed because of changes between major releases (for instance 1.11 XMLs could not be imported in 1.10).

XMLs are exchanged at runtime between some components of the HPC software stack (for instance the resource managers and MPI processes). Building all these components on the same (cluster-wide) hwloc installation is a good way to avoid such incompatibilities.

### 16.5.5 Are synthetic strings compatible between hwloc releases?

Synthetic strings (see Synthetic topologies) are forward-compatible: a synthetic string generated by a release may be imported by future hwloc libraries.

However they are often not backward-compatible because new details may have been added to synthetic descriptions in recent releases. Some flags may be given to hwloc_topology_export_synthetic() to avoid such details and stay backward compatible.

### 16.5.6 Is it possible to share a shared-memory topology between different hwloc releases?

Shared-memory topologies (see Sharing topologies between processes) have strong requirements on compatibility between hwloc libraries. Adopting a shared-memory topology fails if it was exported by a non-compatible hwloc release. Releases with same major revision are usually compatible (e.g. hwloc 2.0.4 may adopt a topology exported by 2.0.3) but different major revisions may be incompatible (e.g. hwloc 2.1.0 cannot adopt from 2.0.x).

Topologies are shared at runtime between some components of the HPC software stack (for instance the resource managers and MPI processes). Building all these components on the same (system-wide) hwloc installation is a good way to avoid such incompatibilities.

# Chapter 17

# Upgrading to the hwloc 2.0 API

See Compatibility between hwloc versions for detecting the hwloc version that you are compiling and/or running against.

## 17.1 New Organization of NUMA nodes and Memory

### 17.1.1 Memory children

In hwloc v1.x, NUMA nodes were inside the tree, for instance Packages contained 2 NUMA nodes which contained a L3 and several cache.

Starting with hwloc v2.0, NUMA nodes are not in the main tree anymore. They are attached under objects as *Memory Children* on the side of normal children. This memory children list starts at `obj->memory_first_child` and its size is `obj->memory_arity`. Hence there can now exist two local NUMA nodes, for instance on Intel Xeon Phi processors.

The normal list of children (starting at `obj->first_child`, ending at `obj->last_child`, of size `obj->arity`, and available as the array `obj->children`) now only contains CPU-side objects: PUs, Cores, Packages, Caches, Groups, Machine and System. hwloc_get_next_child() may still be used to iterate over all children of all lists.

Hence the CPU-side hierarchy is built using normal children, while memory is attached to that hierarchy depending on its affinity.

### 17.1.2 Examples

- a UMA machine with 2 packages and a single NUMA node is now modeled as a "Machine" object with two "Package" children and one "NUMANode" memory children (displayed first in lstopo below):

```
Machine (1024MB total)
  NUMANode L#0 (P#0 1024MB)
  Package L#0
    Core L#0 + PU L#0 (P#0)
    Core L#1 + PU L#1 (P#1)
  Package L#1
    Core L#2 + PU L#2 (P#2)
    Core L#3 + PU L#3 (P#3)
```

- a machine with 2 packages with one NUMA node and 2 cores in each is now:

```
Machine (2048MB total)
  Package L#0
    NUMANode L#0 (P#0 1024MB)
    Core L#0 + PU L#0 (P#0)
    Core L#1 + PU L#1 (P#1)
  Package L#1
    NUMANode L#1 (P#1 1024MB)
    Core L#2 + PU L#2 (P#2)
    Core L#3 + PU L#3 (P#3)
```

- if there are two NUMA nodes per package, a Group object may be added to keep cores together with their local NUMA node:

```
Machine (4096MB total)
  Package L#0
    Group0 L#0
      NUMANode L#0 (P#0 1024MB)
      Core L#0 + PU L#0 (P#0)
      Core L#1 + PU L#1 (P#1)
```

```
    Group0 L#1
      NUMANode L#1 (P#1 1024MB)
      Core L#2 + PU L#2 (P#2)
      Core L#3 + PU L#3 (P#3)
  Package L#1
    [...]
```

- if the platform has L3 caches whose localities are identical to NUMA nodes, Groups aren't needed:

```
Machine (4096MB total)
  Package L#0
    L3 L#0 (16MB)
      NUMANode L#0 (P#0 1024MB)
      Core L#0 + PU L#0 (P#0)
      Core L#1 + PU L#1 (P#1)
    L3 L#1 (16MB)
      NUMANode L#1 (P#1 1024MB)
      Core L#2 + PU L#2 (P#2)
      Core L#3 + PU L#3 (P#3)
  Package L#1
    [...]
```

### 17.1.3 NUMA level and depth

NUMA nodes are not in "main" tree of normal objects anymore. Hence, they don't have a meaningful depth anymore (like I/O and Misc objects). They have a virtual (negative) depth (HWLOC_TYPE_DEPTH_NUMANODE) so that functions manipulating depths and level still work, and so that we can still iterate over the level of NUMA nodes just like for any other level.

For instance we can still use lines such as

```
int depth = hwloc_get_type_depth(topology, HWLOC_OBJ_NUMANODE);
hwloc_obj_t obj = hwloc_get_obj_by_type(topology, HWLOC_OBJ_NUMANODE, 4);
hwloc_obj_t node = hwloc_get_next_obj_by_depth(topology, HWLOC_TYPE_DEPTH_NUMANODE, prev);
```

The NUMA depth should not be compared with others. An unmodified code that still compares NUMA and Package depths (to find out whether Packages contain NUMA or the contrary) would now always assume Packages contain NUMA (because the NUMA depth is negative).

However, the depth of the Normal parents of NUMA nodes may be used instead. In the last example above, NUMA nodes are attached to L3 caches, hence one may compare the depth of Packages and L3 to find out that NUMA nodes are contained in Packages. This depth of parents may be retrieved with hwloc_get_memory_parents_depth(). However, this function may return HWLOC_TYPE_DEPTH_MULTIPLE on future platforms if NUMA nodes are attached to different levels.

### 17.1.4 Finding Local NUMA nodes and looking at Children and Parents

Applications that walked up/down to find NUMANode parent/children must now be updated. Instead of looking directly for a NUMA node, one should now look for an object that has some memory children. NUMA node(s) will be be attached there. For instance, when looking for a NUMA node above a given core `core`:

```
hwloc_obj_t parent = core->parent;
while (parent && !parent->memory_arity)
  parent = parent->parent; /* no memory child, walk up */
if (parent)
  /* use parent->memory_first_child (and its siblings if there are multiple local NUMA nodes) */
```

The list of local NUMA nodes (usually a single one) is also described by the `nodeset` attribute of each object (which contains the physical indexes of these nodes). Iterating over the NUMA level is also an easy way to find local NUMA nodes:

```
hwloc_obj_t tmp = NULL;
while ((tmp = hwloc_get_next_obj_by_type(topology, HWLOC_OBJ_NUMANODE, tmp)) != NULL) {
  if (hwloc_bitmap_isset(obj->nodeset, tmp->os_index))
    /* tmp is a NUMA node local to obj, use it */
}
```

Similarly finding objects that are close to a given NUMA nodes should be updated too. Instead of looking at the NUMA node parents/children, one should now find a Normal parent above that NUMA node, and then look at its parents/children as usual:

```
hwloc_obj_t tmp = obj->parent;
while (hwloc_obj_type_is_memory(tmp))
  tmp = tmp->parent;
/* now use tmp instead of obj */
```

To avoid such hwloc v2.x-specific and NUMA-specific cases in the code, a **generic lookup for any kind of object, including NUMA nodes**, might also be implemented by iterating over a level. For instance finding an object of type `type` which either contains or is included in object `obj` can be performed by traversing the level of that type and comparing CPU sets:

```
hwloc_obj_t tmp = NULL;
while ((tmp = hwloc_get_next_obj_by_type(topology, type, tmp)) != NULL) {
  if (hwloc_bitmap_intersects(tmp->cpuset, obj->cpuset))
    /* tmp matches, use it */
}
```

**This generic lookup works whenever `type` or `obj` are Normal or Memory objects since both have CPU sets. Moreover, it is compatible with the hwloc v1.x API.**

## 17.2 4 Kinds of Objects and Children

### 17.2.1 I/O and Misc children

I/O children are not in the main object children list anymore either. They are in the list starting at `obj->io_first_child` and whose size if `obj->io_arity`.

Misc children are not in the main object children list anymore. They are in the list starting at `obj->misc_first_child` nd whose size if `obj->misc_arity`.

See hwloc_obj for details about children lists.

hwloc_get_next_child() may still be used to iterate over all children of all lists.

### 17.2.2 Kinds of objects

Given the above, objects may now be of 4 kinds:

- Normal (everything not listed below, including Machine, Package, Core, PU, CPU Caches, etc);
- Memory (currently NUMA nodes or Memory-side Caches), attached to parents as Memory children;
- I/O (Bridges, PCI and OS devices), attached to parents as I/O children;
- Misc objects, attached to parents as Misc children.

See hwloc_obj for details about children lists.

For a given object type, the kind may be found with hwloc_obj_type_is_normal(), hwloc_obj_type_is_-memory(), hwloc_obj_type_is_normal(), or comparing with HWLOC_OBJ_MISC.

Normal and Memory objects have (non-NULL) CPU sets and nodesets, while I/O and Misc objects don't have any sets (they are NULL).

## 17.3 HWLOC_OBJ_CACHE replaced

Instead of a single HWLOC_OBJ_CACHE, there are now 8 types HWLOC_OBJ_L1CACHE, ..., HWLOC_OBJ_L5CACHE, HWLOC_OBJ_L1ICACHE, ..., HWLOC_OBJ_L3ICACHE.

Cache object attributes are unchanged.

hwloc_get_cache_type_depth() is not needed to disambiguate cache types anymore since new types can be passed to hwloc_get_type_depth() without ever getting HWLOC_TYPE_DEPTH_MULTIPLE anymore.

hwloc_obj_type_is_cache(), hwloc_obj_type_is_dcache() and hwloc_obj_type_is_icache() may be used to check whether a given type is a cache, data/unified cache or instruction cache.

## 17.4 allowed_cpuset and allowed_nodeset only in the main topology

Objects do not have `allowed_cpuset` and `allowed_nodeset` anymore. They are only available for the entire topology using hwloc_topology_get_allowed_cpuset() and hwloc_topology_get_allowed_-nodeset().

As usual, those are only needed when the INCLUDE_DISALLOWED topology flag is given, which means disallowed objects are kept in the topology. If so, one may find out whether some PUs inside an object is allowed by checking

```
hwloc_bitmap_intersects(obj->cpuset, hwloc_topology_get_allowed_cpuset(topology))
```

Replace cpusets with nodesets for NUMA nodes. To find out which ones, replace intersects() with and() to get the actual intersection.

## 17.5 Object depths are now signed int

`obj->depth` as well as depths given to functions such as hwloc_get_obj_by_depth() or returned by hwloc_topology_get_depth() are now **signed int**.

Other depth such as cache-specific depth attribute are still unsigned.

## 17.6 Memory attributes become NUMANode-specific

Memory attributes such as `obj->memory.local_memory` are now only available in NUMANode-specific attributes in `obj->attr->numanode.local_memory`.

`obj->memory.total_memory` is available in all objects as `obj->total_memory`.

See hwloc_obj_attr_u::hwloc_numanode_attr_s and hwloc_obj for details.

## 17.7 Topology configuration changes

The old ignoring API as well as several configuration flags are replaced with the new filtering API, see hwloc_topology_set_type_filter() and its variants, and hwloc_type_filter_e for details.

- hwloc_topology_ignore_type(), hwloc_topology_ignore_type_keep_structure() and hwloc_topology_ignore_all_keep_structure() are respectively superseded by

```
hwloc_topology_set_type_filter(topology, type, HWLOC_TYPE_FILTER_KEEP_NONE);
hwloc_topology_set_type_filter(topology, type, HWLOC_TYPE_FILTER_KEEP_STRUCTURE);
hwloc_topology_set_all_types_filter(topology, HWLOC_TYPE_FILTER_KEEP_STRUCTURE);
```

  Also, the meaning of KEEP_STRUCTURE has changed (only entire levels may be ignored, instead of single objects), the old behavior is not available anymore.

- HWLOC_TOPOLOGY_FLAG_ICACHES is superseded by

```
hwloc_topology_set_icache_types_filter(topology, HWLOC_TYPE_FILTER_KEEP_ALL);
```

- HWLOC_TOPOLOGY_FLAG_WHOLE_IO, HWLOC_TOPOLOGY_FLAG_IO_DEVICES and HWLOC_TOPOLOGY_FLAG_IO_BRIDGES replaced.

  To keep all I/O devices (PCI, Bridges, and OS devices), use:

```
hwloc_topology_set_io_types_filter(topology, HWLOC_TYPE_FILTER_KEEP_ALL);
```

  To only keep important devices (Bridges with children, common PCI devices and OS devices):

```
hwloc_topology_set_io_types_filter(topology, HWLOC_TYPE_FILTER_KEEP_IMPORTANT);
```

## 17.8 XML changes

2.0 XML files are not compatible with 1.x

2.0 can load 1.x files, but only NUMA distances are imported. Other distance matrices are ignored (they were never used by default anyway).

2.0 can export 1.x-compatible files, but only distances attached to the root object are exported (i.e. distances that cover the entire machine). Other distance matrices are dropped (they were never used by default anyway).

**Users are advised to negociate hwloc versions between exporter and importer:** If the importer isn't 2.x, the exporter should export to 1.x. Otherwise, things should work by default.

Hence hwloc_topology_export_xml() and hwloc_topology_export_xmlbuffer() have a new flags argument. to force a hwloc-1.x-compatible XML export.

- If both always support 2.0, don't pass any flag.
- When the importer uses hwloc 1.x, export with HWLOC_TOPOLOGY_EXPORT_XML_FLAG_V1. Otherwise the importer will fail to import.
- When the exporter uses hwloc 1.x, it cannot pass any flag, and a 2.0 importer can import without problem.

```
#if HWLOC_API_VERSION >= 0x20000
   if (need 1.x compatible XML export)
      hwloc_topology_export_xml(...., HWLOC_TOPOLOGY_EXPORT_XML_FLAG_V1);
   else /* need 2.x compatible XML export */
      hwloc_topology_export_xml(...., 0);
#else
   hwloc_topology_export_xml(....);
#endif
```

Additionally, hwloc_topology_diff_load_xml(), hwloc_topology_diff_load_xmlbuffer(), hwloc_topology_diff_export_xml(), hwloc_topology_diff_export_xmlbuffer() and hwloc_topology_diff_destroy() lost the topology argument: The first argument (topology) isn't needed anymore.

## 17.9 Distances API totally rewritten

The new distances API is in hwloc/distances.h.

Distances are not accessible directly from objects anymore. One should first call hwloc_distances_get() (or a variant) to retrieve distances (possibly with one call to get the number of available distances structures, and another call to actually get them). Then it may consult these structures, and finally release them.

The set of object involved in a distances structure is specified by an array of objects, it may not always cover the entire machine or so.

## 17.10 Return values of functions

Bitmap functions (and a couple other functions) can return errors (in theory).

Most bitmap functions may have to reallocate the internal bitmap storage. In v1.x, they would silently crash if realloc failed. In v2.0, they now return an int that can be negative on error. However, the preallocated storage is 512 bits, hence realloc will not even be used unless you run hwloc on machines with larger PU or NUMAnode indexes.

hwloc_obj_add_info(), hwloc_cpuset_from_nodeset() and hwloc_cpuset_from_nodeset() also return an int, which would be -1 in case of allocation errors.

## 17.11 Misc API changes

- hwloc_type_sscanf() extends hwloc_obj_type_sscanf() by passing a union hwloc_obj_attr_u which may receive Cache, Group, Bridge or OS device attributes.
- hwloc_type_sscanf_as_depth() is also added to directly return the corresponding level depth within a topology.
- hwloc_topology_insert_misc_object_by_cpuset() is replaced with hwloc_topology_alloc_group_object() and hwloc_topology_insert_group_object().
- hwloc_topology_insert_misc_object_by_parent() is replaced with hwloc_topology_insert_misc_object().

## 17.12 API removals and deprecations

- HWLOC_OBJ_SYSTEM removed: The root object is always HWLOC_OBJ_MACHINE
- _membind_nodeset() memory binding interfaces deprecated: One should use the variant without _nodeset suffix and pass the HWLOC_MEMBIND_BYNODESET flag.
- HWLOC_MEMBIND_REPLICATE removed: no supported operating system supports it anymore.
- hwloc_obj_snprintf() removed because it was long-deprecated by hwloc_obj_type_snprintf() and hwloc_obj_attr_snprintf().
- hwloc_obj_type_sscanf() deprecated, hwloc_obj_type_of_string() removed.
- hwloc_cpuset_from/to_nodeset_strict() deprecated: Now useless since all topologies are NUMA. Use the variant without the _strict suffix
- hwloc_distribute() and hwloc_distributev() removed, deprecated by hwloc_distrib().
- The Custom interface (hwloc_topology_set_custom(), etc.) was removed, as well as the corresponding command-line tools (hwloc-assembler, etc.). Topologies always start with object with valid cpusets and nodesets.
- `obj->online_cpuset` removed: Offline PUs are simply listed in the `complete_cpuset` as previously.
- `obj->os_level` removed.

# Chapter 18

# Network Locality (netloc)

Portable abstraction of network topologies for high-performance computing.

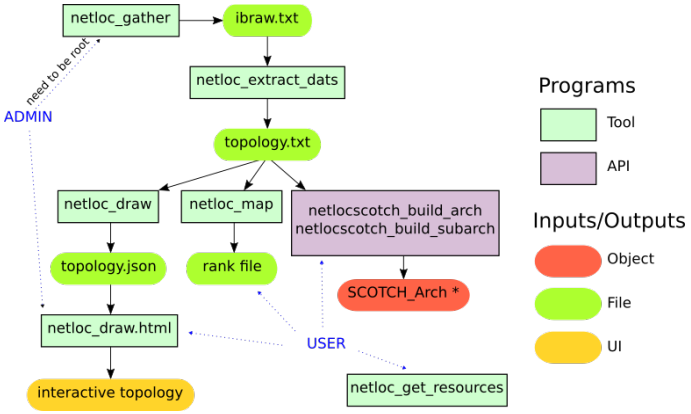
The netloc documentation spans of these sections:

- Network Locality (netloc), this section below
- Netloc with Scotch

## 18.1 Netloc Summary

The Portable Network Locality (netloc) software package provides network topology discovery tools, and an abstract representation of those networks topologies for a range of network types and configurations. It is provided as a companion to the Portable Hardware Locality (hwloc) package. These two software packages work together to provide a comprehensive view of the HPC system topology, spanning from the processor cores in one server to the cores in another - including the complex network(s) in between.

Towards this end, netloc is divided into two sets of components. The first tools are for the admin to extract the information about the topology of the machines with topology discovery tools for each network type and discovery technique (called readers). The second set of tools is for the user to exploit the collected information: to display the topology or create a topology-aware mapping of the processes of an application.




### 18.1.1 Supported Networks

For now, only InfiniBand (See Setup) is supported, but it is planned to be extended it very soon.

## 18.2 Netloc Installation

The generic installation procedure for both hwloc and netloc is described in Installation.

Note that netloc is currently not supported on as many platforms as the original hwloc project. netloc is enabled by default when supported, or can be disabled by passing `\--disable-netloc` to the configure command-line.

## 18.3 Setup

To use Netloc tools, we need two steps. The first step consists in getting information about network directly from tools distributed by manufacturers. For Infiniband, for instance, this operation needs privileges to

access to the network device. For this step we have wrappers in Netloc that will call the right tools with the right options.

The second step will transform the raw files generated by manufacturer tools, into files in a format readable by Netloc tools, and that will not depend on network technologies.

To be clear, let's take an example with Infiniband. This first step is handled by `netloc_ib_gather_raw` that will call `ibnetdiscover` and `ibroutes` tools to generate the necessary raw data files. The step has to be run by an administrator, since the Infiniband tools need to access to the network device.

```
shell$ netloc_ib_gather_raw --help
Usage: netloc_ib_gather_raw [options] <outdir>
  Dumps topology information to <outdir>/ib-raw/
  Subnets are guessed from the <outdir>/hwloc/ directory where
  the hwloc XML exports of some nodes are stored.
Options:
 --sudo
    Pass sudo to internal ibnetdiscover and ibroute invocations.
    Useful when the entire script cannot run as root.
 --hwloc-dir <dir>
    Use <dir> instead of <outdir>/hwloc/ for hwloc XML exports.
 --force-subnet [<subnet>:]<board>:<port> to force the discovery
    Do not guess subnets from hwloc XML exports.
    Force discovery on local board <board> port <port>
    and optionally force the subnet id <subnet>
    instead of reading it from the first GID.
    Examples: --force-subnet mlx4_0:1
              --force-subnet fe80:0000:0000:0000:mlx4_0:1
 --ibnetdiscover /path/to/ibnetdiscover
 --ibroute /path/to/ibroute
    Specify exact location of programs. Default is /usr/bin/<program>
 --sleep <n>
    Sleep for <n> seconds between invocations of programs probing the network
 --ignore-errors
    Ignore errors from ibnetdiscover and ibroute, assume their outputs are ok
 --force -f
    Always rediscover to overwrite existing files without asking
 --verbose -v
    Add verbose messages
 --dry-run
    Do not actually run programs or modify anything
 --help -h
    Show this help

shell$ ./netloc_ib_gather_raw /home/netloc/data
WARNING: Not running as root.
Using /home/netloc/data/hwloc as hwloc lstopo XML directory.

Exporting local node hwloc XML...
  Running lstopo-no-graphics...

Found 1 subnets in hwloc directory:
 Subnet fe80:0000:0000:0000 is locally accessible from board qib0 port 1.

Looking at fe80:0000:0000:0000 (through local board qib0 port 1)...
 Running ibnetdiscover...
 Getting routes...
  Running ibroute for switch 'QLogic 12800-180 GUID=0x00066a00e8001310 L112' LID 18...
  Running ibroute for switch 'QLogic 12800-180 GUID=0x00066a00e8001310 L108' LID 20...
  Running ibroute for switch 'QLogic 12800-180 GUID=0x00066a00e8001310 L102' LID 23...
  Running ibroute for switch 'QLogic 12800-180 GUID=0x00066a00e8001310 L104' LID 25...
  Running ibroute for switch 'QLogic 12800-180 GUID=0x00066a00e8001310 L106' LID 24...
  Running ibroute for switch 'QLogic 12800-180 GUID=0x00066a00e8001310 L114' LID 22...
  Running ibroute for switch 'QLogic 12800-180 GUID=0x00066a00e8001310 L116' LID 21...
  Running ibroute for switch 'QLogic 12800-180 GUID=0x00066a00e8001310 L109' LID 12...
  Running ibroute for switch 'QLogic 12800-180 GUID=0x00066a00e8001310 L111' LID 11...
  Running ibroute for switch 'QLogic 12800-180 GUID=0x00066a00e8001310 L107' LID 13...
```

```
  Running ibroute for switch 'QLogic 12800-180 GUID=0x00066a00e8001310 L103' LID 17...
  Running ibroute for switch 'QLogic 12800-180 GUID=0x00066a00e8001310 L105' LID 16...
  Running ibroute for switch 'QLogic 12800-180 GUID=0x00066a00e8001310 L113' LID 15...
```

The second step, that can be done by a regular user, is done by the tool `netloc_ib_extract_dats`.

```
shell$ netloc_ib_extract_dats --help
Usage: netloc_ib_extract_dats <path to input raw data files> <output path> [--hwloc-dir
<hwloc xml path>]
        hwloc-dir can be an absolute path or a relative path from output path

shell$ netloc_ib_extract_dats /home/netloc/data/ib-raw /home/netloc/data/netloc \
  --hwloc-dir ../hwloc
Read subnet: fe80:0000:0000:0000
2 partitions found
        'node'
        'admin'
```

## 18.4 Topology display

Netloc provides a tool, `netloc_draw.html`, that displays a topology in a web browser, by using a JSON file.

### 18.4.1 Generate the JSON file

In order to display a topology, Netloc needs to generate a JSON file corresponding to a topology. For this operation, the user must run `netloc_draw_to_json`.

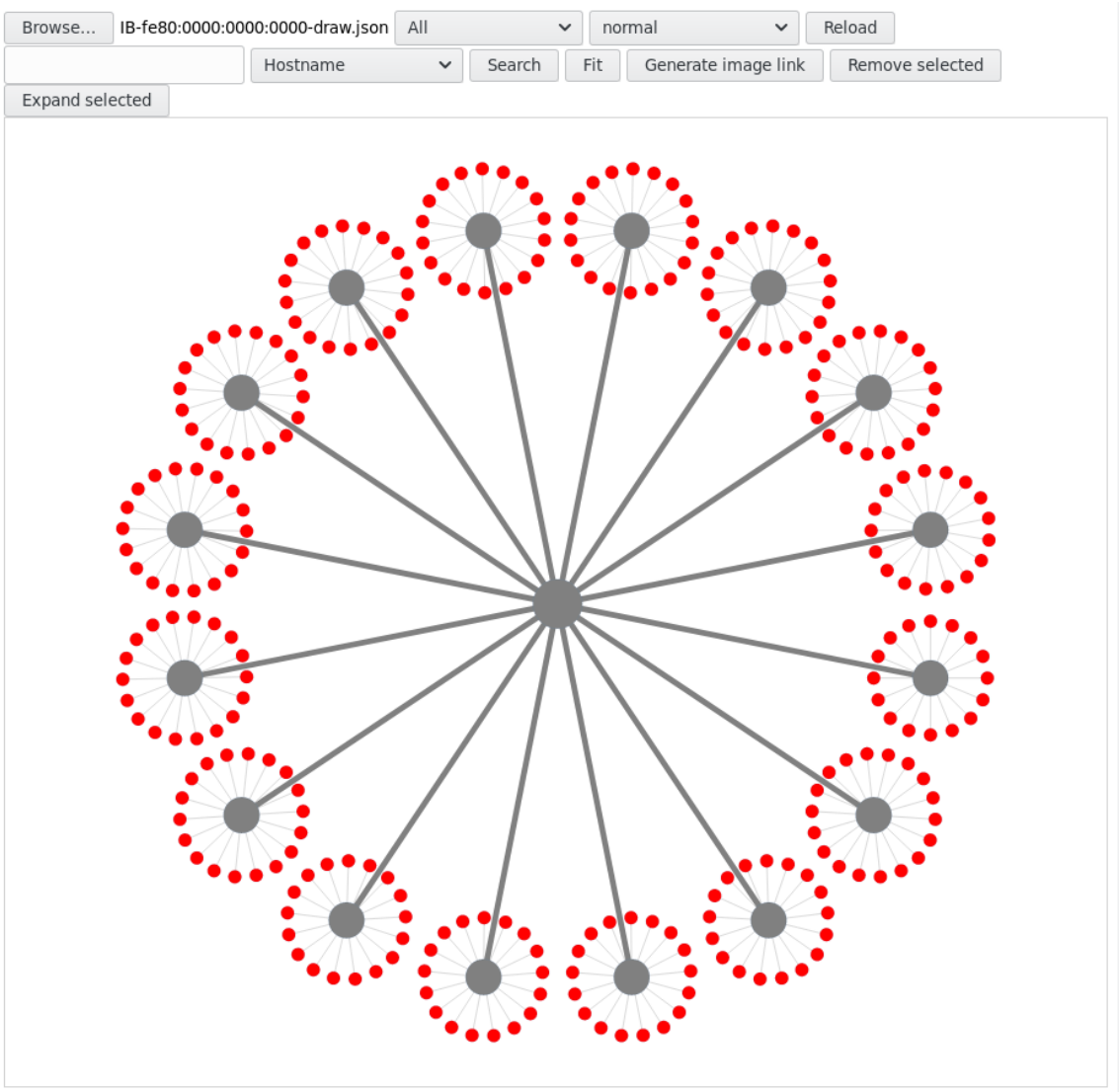
```
shell$ netloc_draw_to_json --help
Usage: netloc_draw_to_json <path to topology directory>

shell$ netloc_draw_to_json /home/netloc/data/netloc
```

The `netloc_draw_to_json` command will write a JSON file for each topology file found in the input directory. The output files, written also in the input directory, can be open by `netloc_draw.html` in a web browser.

### 18.4.2 Using netloc_draw

Once the JSON file is opened, the rendering is generated by the Javascript vis library for computing the position of the nodes. From the interface, it is possible to search for a specific node, to color the nodes, to expand merged switches, to show statistics, to export as an image... The user can interact with the nodes by moving them. For now, there are bugs and other nodes might move too.

The placement of the nodes is done statically if the topology is detected as a tree. If not, vis.js will use physics to find good positions, and it can be very time consuming.


Browse...
IB-fe80:0000:0000:0000-draw.json
All
normal
Reload
Hostname
Search
Fit
Generate image link
Remove selected
Expand selected

# Chapter 19

# Netloc with Scotch

Scotch is a toolbox for graph partitioning [XXX], that can do mapping between a communication graph and an architecture. Netloc interfaces with Scotch, by getting the topology of the machine and building the Scotch architecture. It is also possible to directly build a mapping file that can be given to `mpirun`.

## 19.1 Introduction

Scotch is able to deal architectures to represent the topology of a complete machine. Scotch handles several types of topologies: complete graphs, hypercubes, fat trees, meshes, torus, and random graphs. Moreover, Scotch is able to manage parts of architectures that are called sub-architectures. Thus, from a complete architecture, we can create a sub-architecture that will represent the available resources of the complete machine.

## 19.2 Setup

The first step in order to use Netloc tools is to discover the network. For this task, we provide tools called netloc_gather that are wrappers to the dedicated tools provided by the manufacturer of the network, that generate the raw data given by the devices. This task needs privileges to access to the network devices. Once, this task is completed, the raw data is converted in a generic format independent to the fabric by extract_dats. Figure 1 shows how the different modules of Netloc are linked, and what are the tools provided by Netloc.

## 19.3 Tools and API

When the machine is discovered and all the needed files are generated as seen previously, a user can call the netlocscotch functions from the API and interact with Scotch.

### 19.3.1 Build Scotch architectures

Netloc provides a function to export the built topology into the Scotch format. That will give the possibility to the user to play with the topology in Scotch. Since Netloc matches the discovered topology with known topologies, the Scotch architecture won't be random graphs but known topologies also in Scotch that will lead to optimized graph algorithms. This function is called netlocscotch_build_arch.

When the network topology is a tree, the topology converted by netlocscotch is the complete topology of the machine containing intranode topologies from hwloc. In this case, merging the two levels results in a bigger tree. For other network topologies, the global graph created for Scotch is a generic graph since it not not (at this moment) possible to create nested known architectures.

### 19.3.2 Build Scotch sub-architectures

Most of the time, the user does not have access to the complete machine. He uses a resource manager to run his application and he will gain access only to a set of nodes. In this case getting the Scotch architecture of the complete machine is not relevant. Fortunately, Netloc is also able to build a Scotch sub-architecture that will contain only the available nodes. For this operation the user needs to run a specific program, netloc_get_resources, that will record in a file, the lists of available nodes and available cores by using MPI and hwloc. From this file, the function netlocscotch_build_subarch will build the Scotch sub-architecture.

### 19.3.3 Mapping of processes

A main goal in having all these data about the network topology, especially in Scotch structures, is to help the process placement. For that, we use the mapping of a process graph to the architecture provided by Scotch. As we have seen previously, Netloc is able to detect the structure of the topology and will build the adapted Scotch architecture that will be more efficient than a random structure.

In case, the network topology is not a tree, netlocscotch converts the complete topology into a generic graph. The drawback in that is the Scotch graph algorithms are less efficient. To overcome that, netlocscotch does two steps of mapping: first it maps the processes to the nodes, and then for each node maps the processes to the cores. We have to conduct tests to check if the method gives better results than using a generic graph directly.

The other input needed in Scotch is the process graph. Since we want to optimize the placement to decrease the communication time, a good metric for building the application graph is the amount of communications between all pairs of processes. Studies still have to be done to choose, in the most efficient way, what we take into account to define the amount of communications between the number of messages, the size of messages... This information will be transformed into a process graph.

Once we have a good mapping computed by Scotch, we can give it to the user, or Netloc can even generate the corresponding rank file useful to MPI.

# Chapter 20

# Module Index

## 20.1 Modules

Here is a list of all modules:

# Chapter 21

# Data Structure Index

## 21.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 22

# Module Documentation

## 22.1 API version

### Defines

- #define HWLOC_API_VERSION 0x00020100
- #define HWLOC_COMPONENT_ABI 6

### Functions

- unsigned hwloc_get_api_version (void)

### 22.1.1 Define Documentation

#### 22.1.1.1 #define HWLOC_API_VERSION 0x00020100

Indicate at build time which hwloc API version is being used. This number is updated to (X<<16)+(Y<<8)+Z when a new release X.Y.Z actually modifies the API.

Users may check for available features at build time using this number (see How do I handle API changes?).

**Note:**

This should not be confused with HWLOC_VERSION, the library version. Two stable releases of the same series usually have the same HWLOC_API_VERSION even if their HWLOC_VERSION are different.

#### 22.1.1.2 #define HWLOC_COMPONENT_ABI 6

Current component and plugin ABI version (see hwloc/plugins.h).

### 22.1.2 Function Documentation

#### 22.1.2.1 unsigned hwloc_get_api_version (void)

Indicate at runtime which hwloc API version was used at build time. Should be HWLOC_API_VERSION if running on the same version.

## 22.2 Object Sets (hwloc_cpuset_t and hwloc_nodeset_t)

### Typedefs

- typedef hwloc_bitmap_t hwloc_cpuset_t
- typedef hwloc_const_bitmap_t hwloc_const_cpuset_t
- typedef hwloc_bitmap_t hwloc_nodeset_t
- typedef hwloc_const_bitmap_t hwloc_const_nodeset_t

### 22.2.1 Detailed Description

Hwloc uses bitmaps to represent two distinct kinds of object sets: CPU sets (hwloc_cpuset_t) and NUMA node sets (hwloc_nodeset_t). These types are both typedefs to a common back end type (hwloc_bitmap_t), and therefore all the hwloc bitmap functions are applicable to both hwloc_cpuset_t and hwloc_nodeset_t (see The bitmap API).

The rationale for having two different types is that even though the actions one wants to perform on these types are the same (e.g., enable and disable individual items in the set/mask), they're used in very different contexts: one for specifying which processors to use and one for specifying which NUMA nodes to use. Hence, the name difference is really just to reflect the intent of where the type is used.

### 22.2.2 Typedef Documentation

#### 22.2.2.1 typedef hwloc_const_bitmap_t hwloc_const_cpuset_t

A non-modifiable hwloc_cpuset_t.

#### 22.2.2.2 typedef hwloc_const_bitmap_t hwloc_const_nodeset_t

A non-modifiable hwloc_nodeset_t.

#### 22.2.2.3 typedef hwloc_bitmap_t hwloc_cpuset_t

A CPU set is a bitmap whose bits are set according to CPU physical OS indexes. It may be consulted and modified with the bitmap API as any hwloc_bitmap_t (see hwloc/bitmap.h).

Each bit may be converted into a PU object using hwloc_get_pu_obj_by_os_index().

#### 22.2.2.4 typedef hwloc_bitmap_t hwloc_nodeset_t

A node set is a bitmap whose bits are set according to NUMA memory node physical OS indexes. It may be consulted and modified with the bitmap API as any hwloc_bitmap_t (see hwloc/bitmap.h). Each bit may be converted into a NUMA node object using hwloc_get_numanode_obj_by_os_index().

When binding memory on a system without any NUMA node, the single main memory bank is considered as NUMA node #0.

See also Converting between CPU sets and node sets.

## 22.3 Object Types

### Defines

- #define HWLOC_TYPE_UNORDERED

### Typedefs

- typedef enum hwloc_obj_cache_type_e hwloc_obj_cache_type_t
- typedef enum hwloc_obj_bridge_type_e hwloc_obj_bridge_type_t
- typedef enum hwloc_obj_osdev_type_e hwloc_obj_osdev_type_t

### Enumerations

- enum hwloc_obj_type_t {

  HWLOC_OBJ_MACHINE, HWLOC_OBJ_PACKAGE, HWLOC_OBJ_CORE, HWLOC_OBJ_PU,

  HWLOC_OBJ_L1CACHE, HWLOC_OBJ_L2CACHE, HWLOC_OBJ_L3CACHE, HWLOC_OBJ_L4CACHE,

  HWLOC_OBJ_L5CACHE, HWLOC_OBJ_L1ICACHE, HWLOC_OBJ_L2ICACHE, HWLOC_OBJ_L3ICACHE,

  HWLOC_OBJ_GROUP, HWLOC_OBJ_NUMANODE, HWLOC_OBJ_BRIDGE, HWLOC_OBJ_PCI_DEVICE,

  HWLOC_OBJ_OS_DEVICE, HWLOC_OBJ_MISC, HWLOC_OBJ_MEMCACHE, HWLOC_OBJ_DIE,

  HWLOC_OBJ_TYPE_MAX }
- enum hwloc_obj_cache_type_e { HWLOC_OBJ_CACHE_UNIFIED, HWLOC_OBJ_CACHE_DATA, HWLOC_OBJ_CACHE_INSTRUCTION }
- enum hwloc_obj_bridge_type_e { HWLOC_OBJ_BRIDGE_HOST, HWLOC_OBJ_BRIDGE_PCI }
- enum hwloc_obj_osdev_type_e {

  HWLOC_OBJ_OSDEV_BLOCK, HWLOC_OBJ_OSDEV_GPU, HWLOC_OBJ_OSDEV_NETWORK, HWLOC_OBJ_OSDEV_OPENFABRICS,

  HWLOC_OBJ_OSDEV_DMA, HWLOC_OBJ_OSDEV_COPROC }

### Functions

- int hwloc_compare_types (hwloc_obj_type_t type1, hwloc_obj_type_t type2)

### 22.3.1 Define Documentation

#### 22.3.1.1 #define HWLOC_TYPE_UNORDERED

Value returned by hwloc_compare_types() when types can not be compared.

## 22.3.2 Typedef Documentation

### 22.3.2.1 typedef enum hwloc_obj_bridge_type_e hwloc_obj_bridge_type_t

Type of one side (upstream or downstream) of an I/O bridge.

### 22.3.2.2 typedef enum hwloc_obj_cache_type_e hwloc_obj_cache_type_t

Cache type.

### 22.3.2.3 typedef enum hwloc_obj_osdev_type_e hwloc_obj_osdev_type_t

Type of a OS device.

## 22.3.3 Enumeration Type Documentation

### 22.3.3.1 enum hwloc_obj_bridge_type_e

Type of one side (upstream or downstream) of an I/O bridge.

**Enumerator:**

***HWLOC_OBJ_BRIDGE_HOST*** Host-side of a bridge, only possible upstream.

***HWLOC_OBJ_BRIDGE_PCI*** PCI-side of a bridge.

### 22.3.3.2 enum hwloc_obj_cache_type_e

Cache type.

**Enumerator:**

***HWLOC_OBJ_CACHE_UNIFIED*** Unified cache.

***HWLOC_OBJ_CACHE_DATA*** Data cache.

***HWLOC_OBJ_CACHE_INSTRUCTION*** Instruction cache (filtered out by default).

### 22.3.3.3 enum hwloc_obj_osdev_type_e

Type of a OS device.

**Enumerator:**

***HWLOC_OBJ_OSDEV_BLOCK*** Operating system block device, or non-volatile memory device. For instance "sda" or "dax2.0" on Linux.

***HWLOC_OBJ_OSDEV_GPU*** Operating system GPU device. For instance ":0.0" for a GL display, "card0" for a Linux DRM device.

***HWLOC_OBJ_OSDEV_NETWORK*** Operating system network device. For instance the "eth0" interface on Linux.

***HWLOC_OBJ_OSDEV_OPENFABRICS*** Operating system openfabrics device. For instance the "mlx4_0" InfiniBand HCA, or "hfi1_0" Omni-Path interface on Linux.

***HWLOC_OBJ_OSDEV_DMA*** Operating system dma engine device. For instance the "dma0chan0" DMA channel on Linux.

***HWLOC_OBJ_OSDEV_COPROC*** Operating system co-processor device. For instance "mic0" for a Xeon Phi (MIC) on Linux, "opencl0d0" for a OpenCL device, "cuda0" for a CUDA device.

#### 22.3.3.4 enum hwloc_obj_type_t

Type of topology object.

**Note:**

Do not rely on the ordering or completeness of the values as new ones may be defined in the future! If you need to compare types, use hwloc_compare_types() instead.

**Enumerator:**

***HWLOC_OBJ_MACHINE*** Machine. A set of processors and memory with cache coherency. This type is always used for the root object of a topology, and never used anywhere else. Hence its parent is always `NULL`.

***HWLOC_OBJ_PACKAGE*** Physical package. The physical package that usually gets inserted into a socket on the motherboard. A processor package usually contains multiple cores, and possibly some dies.

***HWLOC_OBJ_CORE*** Core. A computation unit (may be shared by several logical processors).

***HWLOC_OBJ_PU*** Processing Unit, or (Logical) Processor. An execution unit (may share a core with some other logical processors, e.g. in the case of an SMT core). This is the smallest object representing CPU resources, it cannot have any child except Misc objects.

Objects of this kind are always reported and can thus be used as fallback when others are not.

***HWLOC_OBJ_L1CACHE*** Level 1 Data (or Unified) Cache.

***HWLOC_OBJ_L2CACHE*** Level 2 Data (or Unified) Cache.

***HWLOC_OBJ_L3CACHE*** Level 3 Data (or Unified) Cache.

***HWLOC_OBJ_L4CACHE*** Level 4 Data (or Unified) Cache.

***HWLOC_OBJ_L5CACHE*** Level 5 Data (or Unified) Cache.

***HWLOC_OBJ_L1ICACHE*** Level 1 instruction Cache (filtered out by default).

***HWLOC_OBJ_L2ICACHE*** Level 2 instruction Cache (filtered out by default).

***HWLOC_OBJ_L3ICACHE*** Level 3 instruction Cache (filtered out by default).

***HWLOC_OBJ_GROUP*** Group objects. Objects which do not fit in the above but are detected by hwloc and are useful to take into account for affinity. For instance, some operating systems expose their arbitrary processors aggregation this way. And hwloc may insert such objects to group NUMA nodes according to their distances. See also What are these Group objects in my topology?. These objects are removed when they do not bring any structure (see HWLOC_-TYPE_FILTER_KEEP_STRUCTURE).

***HWLOC_OBJ_NUMANODE*** NUMA node. An object that contains memory that is directly and byte-accessible to the host processors. It is usually close to some cores (the corresponding objects are descendants of the NUMA node object in the hwloc tree). This is the smallest object representing Memory resources, it cannot have any child except Misc objects. However it may have Memory-side cache parents.

There is always at least one such object in the topology even if the machine is not NUMA.

Memory objects are not listed in the main children list, but rather in the dedicated Memory children list.

NUMA nodes have a special depth HWLOC_TYPE_DEPTH_NUMANODE instead of a normal depth just like other objects in the main tree.

***HWLOC_OBJ_BRIDGE*** Bridge (filtered out by default). Any bridge (or PCI switch) that connects the host or an I/O bus, to another I/O bus. They are not added to the topology unless I/O discovery is enabled with hwloc_topology_set_flags(). I/O objects are not listed in the main children list, but rather in the dedicated io children list. I/O objects have NULL CPU and node sets.

***HWLOC_OBJ_PCI_DEVICE*** PCI device (filtered out by default). They are not added to the topology unless I/O discovery is enabled with hwloc_topology_set_flags(). I/O objects are not listed in the main children list, but rather in the dedicated io children list. I/O objects have NULL CPU and node sets.

***HWLOC_OBJ_OS_DEVICE*** Operating system device (filtered out by default). They are not added to the topology unless I/O discovery is enabled with hwloc_topology_set_flags(). I/O objects are not listed in the main children list, but rather in the dedicated io children list. I/O objects have NULL CPU and node sets.

***HWLOC_OBJ_MISC*** Miscellaneous objects (filtered out by default). Objects without particular meaning, that can e.g. be added by the application for its own use, or by hwloc for miscellaneous objects such as MemoryModule (DIMMs). These objects are not listed in the main children list, but rather in the dedicated misc children list. Misc objects may only have Misc objects as children, and those are in the dedicated misc children list as well. Misc objects have NULL CPU and node sets.

***HWLOC_OBJ_MEMCACHE*** Memory-side cache (filtered out by default). A cache in front of a specific NUMA node. This object always has at least one NUMA node as a memory child.

Memory objects are not listed in the main children list, but rather in the dedicated Memory children list.

Memory-side cache have a special depth HWLOC_TYPE_DEPTH_MEMCACHE instead of a normal depth just like other objects in the main tree.

***HWLOC_OBJ_DIE*** Die within a physical package. A subpart of the physical package, that contains multiple cores.

***HWLOC_OBJ_TYPE_MAX*** Sentinel value

## 22.3.4 Function Documentation

### 22.3.4.1 int hwloc_compare_types (hwloc_obj_type_t *type1*, hwloc_obj_type_t *type2*)

Compare the depth of two object types. Types shouldn't be compared as they are, since newer ones may be added in the future. This function returns less than, equal to, or greater than zero respectively if `type1` objects usually include `type2` objects, are the same as `type2` objects, or are included in `type2` objects. If the types can not be compared (because neither is usually contained in the other), HWLOC_TYPE_UNORDERED is returned. Object types containing CPUs can always be compared (usually, a system contains machines which contain nodes which contain packages which contain caches, which contain cores, which contain processors).

**Note:**

HWLOC_OBJ_PU will always be the deepest, while HWLOC_OBJ_MACHINE is always the highest. This does not mean that the actual topology will respect that order: e.g. as of today cores may also contain caches, and packages may also contain nodes. This is thus just to be seen as a fallback comparison method.

## 22.4 Object Structure and Attributes

### Data Structures

- struct hwloc_obj

  *Structure of a topology object.*

- union hwloc_obj_attr_u

  *Object type-specific Attributes.*

- struct hwloc_info_s

  *Object info.*

### Typedefs

- typedef struct hwloc_obj * hwloc_obj_t

### 22.4.1 Typedef Documentation

#### 22.4.1.1 typedef struct hwloc_obj* hwloc_obj_t

Convenience typedef; a pointer to a struct hwloc_obj.

## 22.5 Topology Creation and Destruction

### Typedefs

- typedef struct hwloc_topology ∗ hwloc_topology_t

### Functions

- int hwloc_topology_init (hwloc_topology_t ∗topologyp)
- int hwloc_topology_load (hwloc_topology_t topology)
- void hwloc_topology_destroy (hwloc_topology_t topology)
- int hwloc_topology_dup (hwloc_topology_t ∗newtopology, hwloc_topology_t oldtopology)
- int hwloc_topology_abi_check (hwloc_topology_t topology)
- void hwloc_topology_check (hwloc_topology_t topology)

### 22.5.1 Typedef Documentation

#### 22.5.1.1 typedef struct hwloc_topology∗ hwloc_topology_t

Topology context. To be initialized with hwloc_topology_init() and built with hwloc_topology_load().

### 22.5.2 Function Documentation

#### 22.5.2.1 int hwloc_topology_abi_check (hwloc_topology_t *topology*)

Verify that the topology is compatible with the current hwloc library. This is useful when using the same topology structure (in memory) in different libraries that may use different hwloc installations (for instance if one library embeds a specific version of hwloc, while another library uses a default system-wide hwloc installation).

If all libraries/programs use the same hwloc installation, this function always returns success.

**Returns:**

`0` on success.
`-1` with `errno` set to `EINVAL` if incompatible.

**Note:**

If sharing between processes with hwloc_shmem_topology_write(), the relevant check is already performed inside hwloc_shmem_topology_adopt().

#### 22.5.2.2 void hwloc_topology_check (hwloc_topology_t *topology*)

Run internal checks on a topology structure. The program aborts if an inconsistency is detected in the given topology.

**Parameters:**

***topology*** is the topology to be checked

**Note:**

This routine is only useful to developers.
The input topology should have been previously loaded with hwloc_topology_load().

#### 22.5.2.3 void hwloc_topology_destroy (hwloc_topology_t *topology*)

Terminate and free a topology context.

**Parameters:**

***topology*** is the topology to be freed

#### 22.5.2.4 int hwloc_topology_dup (hwloc_topology_t * *newtopology*, hwloc_topology_t *oldtopology*)

Duplicate a topology. The entire topology structure as well as its objects are duplicated into a new one.

This is useful for keeping a backup while modifying a topology.

**Note:**

Object userdata is not duplicated since hwloc does not know what it point to. The objects of both old and new topologies will point to the same userdata.

#### 22.5.2.5 int hwloc_topology_init (hwloc_topology_t * *topologyp*)

Allocate a topology context.

**Parameters:**

→ ***topologyp*** is assigned a pointer to the new allocated context.

**Returns:**

0 on success, -1 on error.

#### 22.5.2.6 int hwloc_topology_load (hwloc_topology_t *topology*)

Build the actual topology. Build the actual topology once initialized with hwloc_topology_init() and tuned with Topology Detection Configuration and Query and Changing the Source of Topology Discovery routines. No other routine may be called earlier using this topology context.

**Parameters:**

***topology*** is the topology to be loaded with objects.

**Returns:**

0 on success, -1 on error.

**Note:**

On failure, the topology is reinitialized. It should be either destroyed with hwloc_topology_destroy() or configured and loaded again.
This function may be called only once per topology.
The binding of the current thread or process may temporarily change during this call but it will be restored before it returns.

**See also:**

Topology Detection Configuration and Query and Changing the Source of Topology Discovery

## 22.6 Object levels, depths and types

### Enumerations

- enum hwloc_get_type_depth_e {
  HWLOC_TYPE_DEPTH_UNKNOWN, HWLOC_TYPE_DEPTH_MULTIPLE, HWLOC_TYPE_DEPTH_NUMANODE, HWLOC_TYPE_DEPTH_BRIDGE,
  HWLOC_TYPE_DEPTH_PCI_DEVICE, HWLOC_TYPE_DEPTH_OS_DEVICE, HWLOC_TYPE_DEPTH_MISC, HWLOC_TYPE_DEPTH_MEMCACHE }

### Functions

- int hwloc_topology_get_depth (hwloc_topology_t restrict topology)
- int hwloc_get_type_depth (hwloc_topology_t topology, hwloc_obj_type_t type)
- int hwloc_get_memory_parents_depth (hwloc_topology_t topology)
- static int hwloc_get_type_or_below_depth (hwloc_topology_t topology, hwloc_obj_type_t type)
- static int hwloc_get_type_or_above_depth (hwloc_topology_t topology, hwloc_obj_type_t type)
- hwloc_obj_type_t hwloc_get_depth_type (hwloc_topology_t topology, int depth)
- unsigned hwloc_get_nbobjs_by_depth (hwloc_topology_t topology, int depth)
- static int hwloc_get_nbobjs_by_type (hwloc_topology_t topology, hwloc_obj_type_t type)
- static hwloc_obj_t hwloc_get_root_obj (hwloc_topology_t topology)
- hwloc_obj_t hwloc_get_obj_by_depth (hwloc_topology_t topology, int depth, unsigned idx)
- static hwloc_obj_t hwloc_get_obj_by_type (hwloc_topology_t topology, hwloc_obj_type_t type, unsigned idx)
- static hwloc_obj_t hwloc_get_next_obj_by_depth (hwloc_topology_t topology, int depth, hwloc_obj_t prev)
- static hwloc_obj_t hwloc_get_next_obj_by_type (hwloc_topology_t topology, hwloc_obj_type_t type, hwloc_obj_t prev)

### 22.6.1 Detailed Description

Be sure to see the figure in Terms and Definitions that shows a complete topology tree, including depths, child/sibling/cousin relationships, and an example of an asymmetric topology where one package has fewer caches than its peers.

### 22.6.2 Enumeration Type Documentation

#### 22.6.2.1 enum hwloc_get_type_depth_e

**Enumerator:**

***HWLOC_TYPE_DEPTH_UNKNOWN*** No object of given type exists in the topology.

***HWLOC_TYPE_DEPTH_MULTIPLE*** Objects of given type exist at different depth in the topology (only for Groups).

***HWLOC_TYPE_DEPTH_NUMANODE*** Virtual depth for NUMA nodes.

***HWLOC_TYPE_DEPTH_BRIDGE*** Virtual depth for bridge object level.

***HWLOC_TYPE_DEPTH_PCI_DEVICE*** Virtual depth for PCI device object level.

***HWLOC_TYPE_DEPTH_OS_DEVICE*** Virtual depth for software device object level.

***HWLOC_TYPE_DEPTH_MISC*** Virtual depth for Misc object.

***HWLOC_TYPE_DEPTH_MEMCACHE*** Virtual depth for MemCache object.

### 22.6.3 Function Documentation

#### 22.6.3.1 hwloc_obj_type_t hwloc_get_depth_type (hwloc_topology_t *topology*, int *depth*)

Returns the type of objects at depth `depth`. `depth` should between 0 and hwloc_topology_get_depth()-1, or a virtual depth such as HWLOC_TYPE_DEPTH_NUMANODE.

**Returns:**

(hwloc_obj_type_t)-1 if depth `depth` does not exist.

#### 22.6.3.2 int hwloc_get_memory_parents_depth (hwloc_topology_t *topology*)

Return the depth of parents where memory objects are attached. Memory objects have virtual negative depths because they are not part of the main CPU-side hierarchy of objects. This depth should not be compared with other level depths.

If all Memory objects are attached to Normal parents at the same depth, this parent depth may be compared to other as usual, for instance for knowing whether NUMA nodes is attached above or below Packages.

**Returns:**

The depth of Normal parents of all memory children if all these parents have the same depth. For instance the depth of the Package level if all NUMA nodes are attached to Package objects.
HWLOC_TYPE_DEPTH_MULTIPLE if Normal parents of all memory children do not have the same depth. For instance if some NUMA nodes are attached to Packages while others are attached to Groups.

#### 22.6.3.3 unsigned hwloc_get_nbobjs_by_depth (hwloc_topology_t *topology*, int *depth*)

Returns the width of level at depth `depth`.

#### 22.6.3.4 static int hwloc_get_nbobjs_by_type (hwloc_topology_t *topology*, hwloc_obj_type_t *type*) [inline, static]

Returns the width of level type `type`. If no object for that type exists, 0 is returned. If there are several levels with objects of that type, -1 is returned.

#### 22.6.3.5 static hwloc_obj_t hwloc_get_next_obj_by_depth (hwloc_topology_t *topology*, int *depth*, hwloc_obj_t *prev*) [inline, static]

Returns the next object at depth `depth`. If `prev` is `NULL`, return the first object at depth `depth`.

#### 22.6.3.6 static hwloc_obj_t hwloc_get_next_obj_by_type (hwloc_topology_t *topology*, hwloc_obj_type_t *type*, hwloc_obj_t *prev*) [inline, static]

Returns the next object of type `type`. If `prev` is `NULL`, return the first object at type `type`. If there are multiple or no depth for given type, return `NULL` and let the caller fallback to hwloc_get_next_obj_by_depth().

#### 22.6.3.7 hwloc_obj_t hwloc_get_obj_by_depth (hwloc_topology_t *topology*, int *depth*, unsigned *idx*)

Returns the topology object at logical index `idx` from depth `depth`.

#### 22.6.3.8 static hwloc_obj_t hwloc_get_obj_by_type (hwloc_topology_t *topology*, hwloc_obj_type_t *type*, unsigned *idx*) [inline, static]

Returns the topology object at logical index `idx` with type `type`. If no object for that type exists, `NULL` is returned. If there are several levels with objects of that type (HWLOC_OBJ_GROUP), `NULL` is returned and the caller may fallback to hwloc_get_obj_by_depth().

#### 22.6.3.9 static hwloc_obj_t hwloc_get_root_obj (hwloc_topology_t *topology*) [inline, static]

Returns the top-object of the topology-tree. Its type is HWLOC_OBJ_MACHINE.

#### 22.6.3.10 int hwloc_get_type_depth (hwloc_topology_t *topology*, hwloc_obj_type_t *type*)

Returns the depth of objects of type `type`. If no object of this type is present on the underlying architecture, or if the OS doesn't provide this kind of information, the function returns HWLOC_TYPE_DEPTH_UNKNOWN.

If type is absent but a similar type is acceptable, see also hwloc_get_type_or_below_depth() and hwloc_get_type_or_above_depth().

If HWLOC_OBJ_GROUP is given, the function may return HWLOC_TYPE_DEPTH_MULTIPLE if multiple levels of Groups exist.

If a NUMA node, I/O or Misc object type is given, the function returns a virtual value because these objects are stored in special levels that are not CPU-related. This virtual depth may be passed to other hwloc functions such as hwloc_get_obj_by_depth() but it should not be considered as an actual depth by the application. In particular, it should not be compared with any other object depth or with the entire topology depth.

**See also:**

hwloc_get_memory_parents_depth().
hwloc_type_sscanf_as_depth() for returning the depth of objects whose type is given as a string.

#### 22.6.3.11 static int hwloc_get_type_or_above_depth (hwloc_topology_t *topology*, hwloc_obj_type_t *type*) [inline, static]

Returns the depth of objects of type `type` or above. If no object of this type is present on the underlying architecture, the function returns the depth of the first "present" object typically containing `type`.

This function is only meaningful for normal object types. If a memory, I/O or Misc object type is given, the corresponding virtual depth is always returned (see hwloc_get_type_depth()).

May return HWLOC_TYPE_DEPTH_MULTIPLE for HWLOC_OBJ_GROUP just like hwloc_get_type_depth().

#### 22.6.3.12 static int hwloc_get_type_or_below_depth (hwloc_topology_t *topology*, hwloc_obj_type_t *type*) [inline, static]

Returns the depth of objects of type `type` or below. If no object of this type is present on the underlying architecture, the function returns the depth of the first "present" object typically found inside `type`.

This function is only meaningful for normal object types. If a memory, I/O or Misc object type is given, the corresponding virtual depth is always returned (see hwloc_get_type_depth()).

May return HWLOC_TYPE_DEPTH_MULTIPLE for HWLOC_OBJ_GROUP just like hwloc_get_type_depth().

#### 22.6.3.13 int hwloc_topology_get_depth (hwloc_topology_t restrict *topology*)

Get the depth of the hierarchical tree of objects. This is the depth of HWLOC_OBJ_PU objects plus one.

**Note:**

NUMA nodes, I/O and Misc objects are ignored when computing the depth of the tree (they are placed on special levels).

## 22.7 Converting between Object Types and Attributes, and Strings

### Functions

- const char * hwloc_obj_type_string (hwloc_obj_type_t type)
- int hwloc_obj_type_snprintf (char *restrict string, size_t size, hwloc_obj_t obj, int verbose)
- int hwloc_obj_attr_snprintf (char *restrict string, size_t size, hwloc_obj_t obj, const char *restrict separator, int verbose)
- int hwloc_type_sscanf (const char *string, hwloc_obj_type_t *typep, union hwloc_obj_attr_u *attrp, size_t attrsize)
- int hwloc_type_sscanf_as_depth (const char *string, hwloc_obj_type_t *typep, hwloc_topology_t topology, int *depthp)

### 22.7.1 Function Documentation

#### 22.7.1.1 int hwloc_obj_attr_snprintf (char *restrict *string*, size_t *size*, hwloc_obj_t *obj*, const char *restrict *separator*, int *verbose*)

Stringify the attributes of a given topology object into a human-readable form. Attribute values are separated by `separator`.

Only the major attributes are printed in non-verbose mode.

If `size` is 0, `string` may safely be `NULL`.

**Returns:**

the number of character that were actually written if not truncating, or that would have been written (not including the ending \0).

#### 22.7.1.2 int hwloc_obj_type_snprintf (char *restrict *string*, size_t *size*, hwloc_obj_t *obj*, int *verbose*)

Stringify the type of a given topology object into a human-readable form. Contrary to hwloc_obj_type_string(), this function includes object-specific attributes (such as the Group depth, the Bridge type, or OS device type) in the output, and it requires the caller to provide the output buffer.

The output is guaranteed to be the same for all objects of a same topology level.

If `verbose` is 1, longer type names are used, e.g. L1Cache instead of L1.

The output string may be parsed back by hwloc_type_sscanf().

If `size` is 0, `string` may safely be `NULL`.

**Returns:**

the number of character that were actually written if not truncating, or that would have been written (not including the ending \0).

#### 22.7.1.3 const char* hwloc_obj_type_string (hwloc_obj_type_t *type*)

Return a constant stringified object type. This function is the basic way to convert a generic type into a string. The output string may be parsed back by hwloc_type_sscanf().

hwloc_obj_type_snprintf() may return a more precise output for a specific object, but it requires the caller to provide the output buffer.

#### 22.7.1.4 int hwloc_type_sscanf (const char * *string*, hwloc_obj_type_t * *typep*, union hwloc_obj_attr_u * *attrp*, size_t *attrsize*)

Return an object type and attributes from a type string. Convert strings such as "Package" or "L1iCache" into the corresponding types. Matching is case-insensitive, and only the first letters are actually required to match.

The matched object type is set in `typep` (which cannot be `NULL`).

Type-specific attributes, for instance Cache type, Cache depth, Group depth, Bridge type or OS Device type may be returned in `attrp`. Attributes that are not specified in the string (for instance "Group" without a depth, or "L2Cache" without a cache type) are set to -1.

`attrp` is only filled if not `NULL` and if its size specified in `attrsize` is large enough. It should be at least as large as union hwloc_obj_attr_u.

**Returns:**

0 if a type was correctly identified, otherwise -1.

**Note:**

This function is guaranteed to match any string returned by hwloc_obj_type_string() or hwloc_obj_type_snprintf().
This is an extended version of the now deprecated hwloc_obj_type_sscanf().

#### 22.7.1.5 int hwloc_type_sscanf_as_depth (const char * *string*, hwloc_obj_type_t * *typep*, hwloc_topology_t *topology*, int * *depthp*)

Return an object type and its level depth from a type string. Convert strings such as "Package" or "L1iCache" into the corresponding types and return in `depthp` the depth of the corresponding level in the topology `topology`.

If no object of this type is present on the underlying architecture, HWLOC_TYPE_DEPTH_UNKNOWN is returned.

If multiple such levels exist (for instance if giving Group without any depth), the function may return HWLOC_TYPE_DEPTH_MULTIPLE instead.

The matched object type is set in `typep` if `typep` is non `NULL`.

**Note:**

This function is similar to hwloc_type_sscanf() followed by hwloc_get_type_depth() but it also automatically disambiguates multiple group levels etc.
This function is guaranteed to match any string returned by hwloc_obj_type_string() or hwloc_obj_type_snprintf().

## 22.8 Consulting and Adding Key-Value Info Attributes

### Functions

- static const char * hwloc_obj_get_info_by_name (hwloc_obj_t obj, const char *name)
- int hwloc_obj_add_info (hwloc_obj_t obj, const char *name, const char *value)

### 22.8.1 Function Documentation

#### 22.8.1.1 int hwloc_obj_add_info (hwloc_obj_t *obj*, const char * *name*, const char * *value*)

Add the given info name and value pair to the given object. The info is appended to the existing info array even if another key with the same name already exists.

The input strings are copied before being added in the object infos.

**Returns:**

`0` on success, `-1` on error.

**Note:**

This function may be used to enforce object colors in the lstopo graphical output by using "lstopoStyle" as a name and "Background=#rrggbb" as a value. See CUSTOM COLORS in the lstopo(1) manpage for details.
If `value` contains some non-printable characters, they will be dropped when exporting to XML, see hwloc_topology_export_xml() in hwloc/export.h.

#### 22.8.1.2 static const char* hwloc_obj_get_info_by_name (hwloc_obj_t *obj*, const char * *name*) [inline, static]

Search the given key name in object infos and return the corresponding value. If multiple keys match the given name, only the first one is returned.

**Returns:**

`NULL` if no such key exists.

# 22.9 CPU binding

## Enumerations

- enum hwloc_cpubind_flags_t { HWLOC_CPUBIND_PROCESS, HWLOC_CPUBIND_THREAD, HWLOC_CPUBIND_STRICT, HWLOC_CPUBIND_NOMEMBIND }

## Functions

- int hwloc_set_cpubind (hwloc_topology_t topology, hwloc_const_cpuset_t set, int flags)
- int hwloc_get_cpubind (hwloc_topology_t topology, hwloc_cpuset_t set, int flags)
- int hwloc_set_proc_cpubind (hwloc_topology_t topology, hwloc_pid_t pid, hwloc_const_cpuset_t set, int flags)
- int hwloc_get_proc_cpubind (hwloc_topology_t topology, hwloc_pid_t pid, hwloc_cpuset_t set, int flags)
- int hwloc_set_thread_cpubind (hwloc_topology_t topology, hwloc_thread_t thread, hwloc_const_cpuset_t set, int flags)
- int hwloc_get_thread_cpubind (hwloc_topology_t topology, hwloc_thread_t thread, hwloc_cpuset_t set, int flags)
- int hwloc_get_last_cpu_location (hwloc_topology_t topology, hwloc_cpuset_t set, int flags)
- int hwloc_get_proc_last_cpu_location (hwloc_topology_t topology, hwloc_pid_t pid, hwloc_cpuset_t set, int flags)

### 22.9.1 Detailed Description

Some operating systems only support binding threads or processes to a single PU. Others allow binding to larger sets such as entire Cores or Packages or even random sets of invididual PUs. In such operating system, the scheduler is free to run the task on one of these PU, then migrate it to another PU, etc. It is often useful to call hwloc_bitmap_singlify() on the target CPU set before passing it to the binding function to avoid these expensive migrations. See the documentation of hwloc_bitmap_singlify() for details.

Some operating systems do not provide all hwloc-supported mechanisms to bind processes, threads, etc. hwloc_topology_get_support() may be used to query about the actual CPU binding support in the currently used operating system.

When the requested binding operation is not available and the HWLOC_CPUBIND_STRICT flag was passed, the function returns -1. `errno` is set to `ENOSYS` when it is not possible to bind the requested kind of object processes/threads. errno is set to `EXDEV` when the requested cpuset can not be enforced (e.g. some systems only allow one CPU, and some other systems only allow one NUMA node).

If HWLOC_CPUBIND_STRICT was not passed, the function may fail as well, or the operating system may use a slightly different operation (with side-effects, smaller binding set, etc.) when the requested operation is not exactly supported.

The most portable version that should be preferred over the others, whenever possible, is the following one which just binds the current program, assuming it is single-threaded:

```
hwloc_set_cpubind(topology, set, 0),
```

If the program may be multithreaded, the following one should be preferred to only bind the current thread:

```
hwloc_set_cpubind(topology, set, HWLOC_CPUBIND_THREAD),
```

**See also:**

Some example codes are available under doc/examples/ in the source tree.

**Note:**

To unbind, just call the binding function with either a full cpuset or a cpuset equal to the system cpuset.
On some operating systems, CPU binding may have effects on memory binding, see HWLOC_-CPUBIND_NOMEMBIND
Running lstopo \--top or hwloc-ps can be a very convenient tool to check how binding actually happened.

### 22.9.2 Enumeration Type Documentation

#### 22.9.2.1 enum hwloc_cpubind_flags_t

Process/Thread binding flags. These bit flags can be used to refine the binding policy.

The default (0) is to bind the current process, assumed to be single-threaded, in a non-strict way. This is the most portable way to bind as all operating systems usually provide it.

**Note:**

Not all systems support all kinds of binding. See the "Detailed Description" section of CPU binding for a description of errors that can occur.

**Enumerator:**

***HWLOC_CPUBIND_PROCESS*** Bind all threads of the current (possibly) multithreaded process.

***HWLOC_CPUBIND_THREAD*** Bind current thread of current process.

***HWLOC_CPUBIND_STRICT*** Request for strict binding from the OS. By default, when the designated CPUs are all busy while other CPUs are idle, operating systems may execute the thread/process on those other CPUs instead of the designated CPUs, to let them progress anyway. Strict binding means that the thread/process will _never_ execute on other cpus than the designated CPUs, even when those are busy with other tasks and other CPUs are idle.

**Note:**

Depending on the operating system, strict binding may not be possible (e.g., the OS does not implement it) or not allowed (e.g., for an administrative reasons), and the function will fail in that case.

When retrieving the binding of a process, this flag checks whether all its threads actually have the same binding. If the flag is not given, the binding of each thread will be accumulated.

**Note:**

This flag is meaningless when retrieving the binding of a thread.

***HWLOC_CPUBIND_NOMEMBIND*** Avoid any effect on memory binding. On some operating systems, some CPU binding function would also bind the memory on the corresponding NUMA node. It is often not a problem for the application, but if it is, setting this flag will make hwloc avoid using OS functions that would also bind memory. This will however reduce the support of CPU bindings, i.e. potentially return -1 with errno set to ENOSYS in some cases.

This flag is only meaningful when used with functions that set the CPU binding. It is ignored when used with functions that get CPU binding information.

### 22.9.3 Function Documentation

#### 22.9.3.1 int hwloc_get_cpubind (hwloc_topology_t *topology*, hwloc_cpuset_t *set*, int *flags*)

Get current process or thread binding. Writes into `set` the physical cpuset which the process or thread (according to *flags*) was last bound to.

#### 22.9.3.2 int hwloc_get_last_cpu_location (hwloc_topology_t *topology*, hwloc_cpuset_t *set*, int *flags*)

Get the last physical CPU where the current process or thread ran. The operating system may move some tasks from one processor to another at any time according to their binding, so this function may return something that is already outdated.

`flags` can include either HWLOC_CPUBIND_PROCESS or HWLOC_CPUBIND_THREAD to specify whether the query should be for the whole process (union of all CPUs on which all threads are running), or only the current thread. If the process is single-threaded, flags can be set to zero to let hwloc use whichever method is available on the underlying OS.

#### 22.9.3.3 int hwloc_get_proc_cpubind (hwloc_topology_t *topology*, hwloc_pid_t *pid*, hwloc_cpuset_t *set*, int *flags*)

Get the current physical binding of process `pid`.

**Note:**

`hwloc_pid_t` is `pid_t` on Unix platforms, and `HANDLE` on native Windows platforms.
As a special case on Linux, if a tid (thread ID) is supplied instead of a pid (process ID) and HWLOC_CPUBIND_THREAD is passed in flags, the binding for that specific thread is returned.
On non-Linux systems, HWLOC_CPUBIND_THREAD can not be used in `flags`.

#### 22.9.3.4 int hwloc_get_proc_last_cpu_location (hwloc_topology_t *topology*, hwloc_pid_t *pid*, hwloc_cpuset_t *set*, int *flags*)

Get the last physical CPU where a process ran. The operating system may move some tasks from one processor to another at any time according to their binding, so this function may return something that is already outdated.

**Note:**

`hwloc_pid_t` is `pid_t` on Unix platforms, and `HANDLE` on native Windows platforms.
As a special case on Linux, if a tid (thread ID) is supplied instead of a pid (process ID) and HWLOC_CPUBIND_THREAD is passed in flags, the last CPU location of that specific thread is returned.
On non-Linux systems, HWLOC_CPUBIND_THREAD can not be used in `flags`.

#### 22.9.3.5 int hwloc_get_thread_cpubind (hwloc_topology_t *topology*, hwloc_thread_t *thread*, hwloc_cpuset_t *set*, int *flags*)

Get the current physical binding of thread `tid`.

**Note:**

hwloc_thread_t is pthread_t on Unix platforms, and HANDLE on native Windows platforms. HWLOC_CPUBIND_PROCESS can not be used in flags.

#### 22.9.3.6 int hwloc_set_cpubind (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*, int *flags*)

Bind current process or thread on cpus given in physical bitmap set.

**Returns:**

-1 with errno set to ENOSYS if the action is not supported
-1 with errno set to EXDEV if the binding cannot be enforced

#### 22.9.3.7 int hwloc_set_proc_cpubind (hwloc_topology_t *topology*, hwloc_pid_t *pid*, hwloc_const_cpuset_t *set*, int *flags*)

Bind a process pid on cpus given in physical bitmap set.

**Note:**

hwloc_pid_t is pid_t on Unix platforms, and HANDLE on native Windows platforms.
As a special case on Linux, if a tid (thread ID) is supplied instead of a pid (process ID) and HWLOC_CPUBIND_THREAD is passed in flags, the binding is applied to that specific thread.
On non-Linux systems, HWLOC_CPUBIND_THREAD can not be used in flags.

#### 22.9.3.8 int hwloc_set_thread_cpubind (hwloc_topology_t *topology*, hwloc_thread_t *thread*, hwloc_const_cpuset_t *set*, int *flags*)

Bind a thread thread on cpus given in physical bitmap set.

**Note:**

hwloc_thread_t is pthread_t on Unix platforms, and HANDLE on native Windows platforms. HWLOC_CPUBIND_PROCESS can not be used in flags.

## 22.10 Memory binding

### Enumerations

- enum hwloc_membind_policy_t {

  HWLOC_MEMBIND_DEFAULT, HWLOC_MEMBIND_FIRSTTOUCH, HWLOC_MEMBIND_BIND, HWLOC_MEMBIND_INTERLEAVE,

  HWLOC_MEMBIND_NEXTTOUCH, HWLOC_MEMBIND_MIXED }
- enum hwloc_membind_flags_t {

  HWLOC_MEMBIND_PROCESS, HWLOC_MEMBIND_THREAD, HWLOC_MEMBIND_STRICT, HWLOC_MEMBIND_MIGRATE,

  HWLOC_MEMBIND_NOCPUBIND, HWLOC_MEMBIND_BYNODESET }

### Functions

- int hwloc_set_membind (hwloc_topology_t topology, hwloc_const_bitmap_t set, hwloc_membind_policy_t policy, int flags)
- int hwloc_get_membind (hwloc_topology_t topology, hwloc_bitmap_t set, hwloc_membind_policy_t *policy, int flags)
- int hwloc_set_proc_membind (hwloc_topology_t topology, hwloc_pid_t pid, hwloc_const_bitmap_t set, hwloc_membind_policy_t policy, int flags)
- int hwloc_get_proc_membind (hwloc_topology_t topology, hwloc_pid_t pid, hwloc_bitmap_t set, hwloc_membind_policy_t *policy, int flags)
- int hwloc_set_area_membind (hwloc_topology_t topology, const void *addr, size_t len, hwloc_const_bitmap_t set, hwloc_membind_policy_t policy, int flags)
- int hwloc_get_area_membind (hwloc_topology_t topology, const void *addr, size_t len, hwloc_bitmap_t set, hwloc_membind_policy_t *policy, int flags)
- int hwloc_get_area_memlocation (hwloc_topology_t topology, const void *addr, size_t len, hwloc_bitmap_t set, int flags)
- void * hwloc_alloc (hwloc_topology_t topology, size_t len)
- void * hwloc_alloc_membind (hwloc_topology_t topology, size_t len, hwloc_const_bitmap_t set, hwloc_membind_policy_t policy, int flags)
- static void * hwloc_alloc_membind_policy (hwloc_topology_t topology, size_t len, hwloc_const_bitmap_t set, hwloc_membind_policy_t policy, int flags)
- int hwloc_free (hwloc_topology_t topology, void *addr, size_t len)

### 22.10.1 Detailed Description

Memory binding can be done three ways:

- explicit memory allocation thanks to hwloc_alloc_membind() and friends: the binding will have effect on the memory allocated by these functions.
- implicit memory binding through binding policy: hwloc_set_membind() and friends only define the current policy of the process, which will be applied to the subsequent calls to malloc() and friends.
- migration of existing memory ranges, thanks to hwloc_set_area_membind() and friends, which move already-allocated data.

Not all operating systems support all three ways. hwloc_topology_get_support() may be used to query about the actual memory binding support in the currently used operating system.

When the requested binding operation is not available and the HWLOC_MEMBIND_STRICT flag was passed, the function returns -1. `errno` will be set to `ENOSYS` when the system does support the specified action or policy (e.g., some systems only allow binding memory on a per-thread basis, whereas other systems only allow binding memory for all threads in a process). `errno` will be set to EXDEV when the requested set can not be enforced (e.g., some systems only allow binding memory to a single NUMA node).

If HWLOC_MEMBIND_STRICT was not passed, the function may fail as well, or the operating system may use a slightly different operation (with side-effects, smaller binding set, etc.) when the requested operation is not exactly supported.

The most portable form that should be preferred over the others whenever possible is as follows. It allocates some memory hopefully bound to the specified set. To do so, hwloc will possibly have to change the current memory binding policy in order to actually get the memory bound, if the OS does not provide any other way to simply allocate bound memory without changing the policy for all allocations. That is the difference with hwloc_alloc_membind(), which will never change the current memory binding policy.

```
hwloc_alloc_membind_policy(topology, size, set,
                           HWLOC_MEMBIND_BIND, 0);
```

Each hwloc memory binding function takes a bitmap argument that is a CPU set by default, or a NUMA memory node set if the flag HWLOC_MEMBIND_BYNODESET is specified. See Object Sets (hwloc_cpuset_t and hwloc_nodeset_t) and The bitmap API for a discussion of CPU sets and NUMA memory node sets. It is also possible to convert between CPU set and node set using hwloc_cpuset_to_nodeset() or hwloc_cpuset_from_nodeset().

Memory binding by CPU set cannot work for CPU-less NUMA memory nodes. Binding by nodeset should therefore be preferred whenever possible.

**See also:**

Some example codes are available under doc/examples/ in the source tree.

**Note:**

On some operating systems, memory binding affects the CPU binding; see HWLOC_MEMBIND_NOCPUBIND

## 22.10.2 Enumeration Type Documentation

### 22.10.2.1 enum hwloc_membind_flags_t

Memory binding flags. These flags can be used to refine the binding policy. All flags can be logically OR'ed together with the exception of HWLOC_MEMBIND_PROCESS and HWLOC_MEMBIND_THREAD; these two flags are mutually exclusive.

Not all systems support all kinds of binding. hwloc_topology_get_support() may be used to query about the actual memory binding support in the currently used operating system. See the "Detailed Description" section of Memory binding for a description of errors that can occur.

**Enumerator:**

***HWLOC_MEMBIND_PROCESS*** Set policy for all threads of the specified (possibly multi-threaded) process. This flag is mutually exclusive with HWLOC_MEMBIND_THREAD.

***HWLOC_MEMBIND_THREAD*** Set policy for a specific thread of the current process. This flag is mutually exclusive with HWLOC_MEMBIND_PROCESS.

***HWLOC_MEMBIND_STRICT*** Request strict binding from the OS. The function will fail if the binding can not be guaranteed / completely enforced.

This flag has slightly different meanings depending on which function it is used with.

***HWLOC_MEMBIND_MIGRATE*** Migrate existing allocated memory. If the memory cannot be migrated and the HWLOC_MEMBIND_STRICT flag is passed, an error will be returned.

***HWLOC_MEMBIND_NOCPUBIND*** Avoid any effect on CPU binding. On some operating systems, some underlying memory binding functions also bind the application to the corresponding CPU(s). Using this flag will cause hwloc to avoid using OS functions that could potentially affect CPU bindings. Note, however, that using NOCPUBIND may reduce hwloc's overall memory binding support. Specifically: some of hwloc's memory binding functions may fail with errno set to ENOSYS when used with NOCPUBIND.

***HWLOC_MEMBIND_BYNODESET*** Consider the bitmap argument as a nodeset. The bitmap argument is considered a nodeset if this flag is given, or a cpuset otherwise by default.

Memory binding by CPU set cannot work for CPU-less NUMA memory nodes. Binding by nodeset should therefore be preferred whenever possible.

#### 22.10.2.2 enum hwloc_membind_policy_t

Memory binding policy. These constants can be used to choose the binding policy. Only one policy can be used at a time (i.e., the values cannot be OR'ed together).

Not all systems support all kinds of binding. hwloc_topology_get_support() may be used to query about the actual memory binding policy support in the currently used operating system. See the "Detailed Description" section of Memory binding for a description of errors that can occur.

**Enumerator:**

***HWLOC_MEMBIND_DEFAULT*** Reset the memory allocation policy to the system default. Depending on the operating system, this may correspond to HWLOC_MEMBIND_FIRSTTOUCH (Linux, FreeBSD), or HWLOC_MEMBIND_BIND (AIX, HP-UX, Solaris, Windows). This policy is never returned by get membind functions. The nodeset argument is ignored.

***HWLOC_MEMBIND_FIRSTTOUCH*** Allocate each memory page individually on the local NUMA node of the thread that touches it. The given nodeset should usually be hwloc_topology_get_topology_nodeset() so that the touching thread may run and allocate on any node in the system.

On AIX, if the nodeset is smaller, pages are allocated locally (if the local node is in the nodeset) or from a random non-local node (otherwise).

***HWLOC_MEMBIND_BIND*** Allocate memory on the specified nodes.

***HWLOC_MEMBIND_INTERLEAVE*** Allocate memory on the given nodes in an interleaved / round-robin manner. The precise layout of the memory across multiple NUMA nodes is OS/system specific. Interleaving can be useful when threads distributed across the specified NUMA nodes will all be accessing the whole memory range concurrently, since the interleave will then balance the memory references.

***HWLOC_MEMBIND_NEXTTOUCH*** For each page bound with this policy, by next time it is touched (and next time only), it is moved from its current location to the local NUMA node of the thread where the memory reference occurred (if it needs to be moved at all).

***HWLOC_MEMBIND_MIXED*** Returned by get_membind() functions when multiple threads or parts of a memory area have differing memory binding policies. Also returned when binding is unknown because binding hooks are empty when the topology is loaded from XML without HWLOC_THISSYSTEM=1, etc.

### 22.10.3 Function Documentation

#### 22.10.3.1 void* hwloc_alloc (hwloc_topology_t *topology*, size_t *len*)

Allocate some memory. This is equivalent to malloc(), except that it tries to allocate page-aligned memory from the OS.

**Note:**

The allocated memory should be freed with hwloc_free().

#### 22.10.3.2 void* hwloc_alloc_membind (hwloc_topology_t *topology*, size_t *len*, hwloc_const_bitmap_t *set*, hwloc_membind_policy_t *policy*, int *flags*)

Allocate some memory on NUMA memory nodes specified by `set`.

**Returns:**

NULL with errno set to ENOSYS if the action is not supported and HWLOC_MEMBIND_STRICT is given
NULL with errno set to EXDEV if the binding cannot be enforced and HWLOC_MEMBIND_STRICT is given
NULL with errno set to ENOMEM if the memory allocation failed even before trying to bind.

If HWLOC_MEMBIND_BYNODESET is specified, set is considered a nodeset. Otherwise it's a cpuset.

**Note:**

The allocated memory should be freed with hwloc_free().

#### 22.10.3.3 static void* hwloc_alloc_membind_policy (hwloc_topology_t *topology*, size_t *len*, hwloc_const_bitmap_t *set*, hwloc_membind_policy_t *policy*, int *flags*) `[inline, static]`

Allocate some memory on NUMA memory nodes specified by `set`. This is similar to hwloc_alloc_membind_nodeset() except that it is allowed to change the current memory binding policy, thus providing more binding support, at the expense of changing the current state.

If HWLOC_MEMBIND_BYNODESET is specified, set is considered a nodeset. Otherwise it's a cpuset.

#### 22.10.3.4 int hwloc_free (hwloc_topology_t *topology*, void * *addr*, size_t *len*)

Free memory that was previously allocated by hwloc_alloc() or hwloc_alloc_membind().

#### 22.10.3.5 int hwloc_get_area_membind (hwloc_topology_t *topology*, const void * *addr*, size_t *len*, hwloc_bitmap_t *set*, hwloc_membind_policy_t * *policy*, int *flags*)

Query the CPUs near the physical NUMA node(s) and binding policy of the memory identified by (`addr`, `len` ). This function has two output parameters: `set` and `policy`. The values returned in these parameters depend on both the `flags` passed in and the memory binding policies and nodesets of the pages in the address range.

If HWLOC_MEMBIND_STRICT is specified, the target pages are first checked to see if they all have the same memory binding policy and nodeset. If they do not, -1 is returned and errno is set to EXDEV. If they are identical across all pages, the set and policy are returned in `set` and `policy`, respectively.

If HWLOC_MEMBIND_STRICT is not specified, the union of all NUMA node(s) containing pages in the address range is calculated. If all pages in the target have the same policy, it is returned in `policy`. Otherwise, `policy` is set to HWLOC_MEMBIND_MIXED.

If HWLOC_MEMBIND_BYNODESET is specified, set is considered a nodeset. Otherwise it's a cpuset.

If any other flags are specified, -1 is returned and errno is set to EINVAL.

If `len` is 0, -1 is returned and errno is set to EINVAL.

#### 22.10.3.6 int hwloc_get_area_memlocation (hwloc_topology_t *topology*, const void * *addr*, size_t *len*, hwloc_bitmap_t *set*, int *flags*)

Get the NUMA nodes where memory identified by (`addr`, `len` ) is physically allocated. Fills `set` according to the NUMA nodes where the memory area pages are physically allocated. If no page is actually allocated yet, `set` may be empty.

If pages spread to multiple nodes, it is not specified whether they spread equitably, or whether most of them are on a single node, etc.

The operating system may move memory pages from one processor to another at any time according to their binding, so this function may return something that is already outdated.

If HWLOC_MEMBIND_BYNODESET is specified in `flags`, set is considered a nodeset. Otherwise it's a cpuset.

If `len` is 0, `set` is emptied.

#### 22.10.3.7 int hwloc_get_membind (hwloc_topology_t *topology*, hwloc_bitmap_t *set*, hwloc_membind_policy_t * *policy*, int *flags*)

Query the default memory binding policy and physical locality of the current process or thread. This function has two output parameters: `set` and `policy`. The values returned in these parameters depend on both the `flags` passed in and the current memory binding policies and nodesets in the queried target.

Passing the HWLOC_MEMBIND_PROCESS flag specifies that the query target is the current policies and nodesets for all the threads in the current process. Passing HWLOC_MEMBIND_THREAD specifies that the query target is the current policy and nodeset for only the thread invoking this function.

If neither of these flags are passed (which is the most portable method), the process is assumed to be single threaded. This allows hwloc to use either process-based OS functions or thread-based OS functions, depending on which are available.

HWLOC_MEMBIND_STRICT is only meaningful when HWLOC_MEMBIND_PROCESS is also specified. In this case, hwloc will check the default memory policies and nodesets for all threads in the process. If they are not identical, -1 is returned and errno is set to EXDEV. If they are identical, the values are returned in `set` and `policy`.

Otherwise, if HWLOC_MEMBIND_PROCESS is specified (and HWLOC_MEMBIND_STRICT is *not* specified), the default set from each thread is logically OR'ed together. If all threads' default policies are the same, `policy` is set to that policy. If they are different, `policy` is set to HWLOC_MEMBIND_MIXED.

In the HWLOC_MEMBIND_THREAD case (or when neither HWLOC_MEMBIND_PROCESS or HWLOC_MEMBIND_THREAD is specified), there is only one set and policy; they are returned in `set`

and policy, respectively.

If HWLOC_MEMBIND_BYNODESET is specified, set is considered a nodeset. Otherwise it's a cpuset.

If any other flags are specified, -1 is returned and errno is set to EINVAL.

#### 22.10.3.8 int hwloc_get_proc_membind (hwloc_topology_t *topology*, hwloc_pid_t *pid*, hwloc_bitmap_t *set*, hwloc_membind_policy_t * *policy*, int *flags*)

Query the default memory binding policy and physical locality of the specified process. This function has two output parameters: set and policy. The values returned in these parameters depend on both the flags passed in and the current memory binding policies and nodesets in the queried target.

Passing the HWLOC_MEMBIND_PROCESS flag specifies that the query target is the current policies and nodesets for all the threads in the specified process. If HWLOC_MEMBIND_PROCESS is not specified (which is the most portable method), the process is assumed to be single threaded. This allows hwloc to use either process-based OS functions or thread-based OS functions, depending on which are available.

Note that it does not make sense to pass HWLOC_MEMBIND_THREAD to this function.

If HWLOC_MEMBIND_STRICT is specified, hwloc will check the default memory policies and nodesets for all threads in the specified process. If they are not identical, -1 is returned and errno is set to EXDEV. If they are identical, the values are returned in set and policy.

Otherwise, set is set to the logical OR of all threads' default set. If all threads' default policies are the same, policy is set to that policy. If they are different, policy is set to HWLOC_MEMBIND_MIXED.

If HWLOC_MEMBIND_BYNODESET is specified, set is considered a nodeset. Otherwise it's a cpuset.

If any other flags are specified, -1 is returned and errno is set to EINVAL.

**Note:**

hwloc_pid_t is pid_t on Unix platforms, and HANDLE on native Windows platforms.

#### 22.10.3.9 int hwloc_set_area_membind (hwloc_topology_t *topology*, const void * *addr*, size_t *len*, hwloc_const_bitmap_t *set*, hwloc_membind_policy_t *policy*, int *flags*)

Bind the already-allocated memory identified by (addr, len) to the NUMA node(s) specified by set. If HWLOC_MEMBIND_BYNODESET is specified, set is considered a nodeset. Otherwise it's a cpuset.

**Returns:**

0 if len is 0.
-1 with errno set to ENOSYS if the action is not supported
-1 with errno set to EXDEV if the binding cannot be enforced

#### 22.10.3.10 int hwloc_set_membind (hwloc_topology_t *topology*, hwloc_const_bitmap_t *set*, hwloc_membind_policy_t *policy*, int *flags*)

Set the default memory binding policy of the current process or thread to prefer the NUMA node(s) specified by set. If neither HWLOC_MEMBIND_PROCESS nor HWLOC_MEMBIND_THREAD is specified, the current process is assumed to be single-threaded. This is the most portable form as it permits hwloc to use either process-based OS functions or thread-based OS functions, depending on which are available.

If HWLOC_MEMBIND_BYNODESET is specified, set is considered a nodeset. Otherwise it's a cpuset.

**Returns:**

-1 with errno set to ENOSYS if the action is not supported
-1 with errno set to EXDEV if the binding cannot be enforced

#### 22.10.3.11 int hwloc_set_proc_membind (hwloc_topology_t *topology*, hwloc_pid_t *pid*, hwloc_const_bitmap_t *set*, hwloc_membind_policy_t *policy*, int *flags*)

Set the default memory binding policy of the specified process to prefer the NUMA node(s) specified by `set`. If HWLOC_MEMBIND_BYNODESET is specified, set is considered a nodeset. Otherwise it's a cpuset.

**Returns:**

-1 with errno set to ENOSYS if the action is not supported
-1 with errno set to EXDEV if the binding cannot be enforced

**Note:**

`hwloc_pid_t` is `pid_t` on Unix platforms, and `HANDLE` on native Windows platforms.

## 22.11 Changing the Source of Topology Discovery

### Enumerations

- enum hwloc_topology_components_flag_e { HWLOC_TOPOLOGY_COMPONENTS_FLAG_BLACKLIST }

### Functions

- int hwloc_topology_set_pid (hwloc_topology_t restrict topology, hwloc_pid_t pid)
- int hwloc_topology_set_synthetic (hwloc_topology_t restrict topology, const char *restrict description)
- int hwloc_topology_set_xml (hwloc_topology_t restrict topology, const char *restrict xmlpath)
- int hwloc_topology_set_xmlbuffer (hwloc_topology_t restrict topology, const char *restrict buffer, int size)
- int hwloc_topology_set_components (hwloc_topology_t restrict topology, unsigned long flags, const char *restrict name)

### 22.11.1 Detailed Description

If none of the functions below is called, the default is to detect all the objects of the machine that the caller is allowed to access.

This default behavior may also be modified through environment variables if the application did not modify it already. Setting HWLOC_XMLFILE in the environment enforces the discovery from a XML file as if hwloc_topology_set_xml() had been called. Setting HWLOC_SYNTHETIC enforces a synthetic topology as if hwloc_topology_set_synthetic() had been called.

Finally, HWLOC_THISSYSTEM enforces the return value of hwloc_topology_is_thissystem().

### 22.11.2 Enumeration Type Documentation

#### 22.11.2.1 enum hwloc_topology_components_flag_e

Flags to be passed to hwloc_topology_set_components().

**Enumerator:**

***HWLOC_TOPOLOGY_COMPONENTS_FLAG_BLACKLIST*** Blacklist the target component from being used.

### 22.11.3 Function Documentation

#### 22.11.3.1 int hwloc_topology_set_components (hwloc_topology_t restrict *topology*, unsigned long *flags*, const char *restrict *name*)

Prevent a discovery component from being used for a topology. `name` is the name of the discovery component that should not be used when loading topology `topology`. The name is a string such as "cuda".

For components with multiple phases, it may also be suffixed with the name of a phase, for instance "linux:io".

`flags` should be HWLOC_TOPOLOGY_COMPONENTS_FLAG_BLACKLIST.

This may be used to avoid expensive parts of the discovery process. For instance, CUDA-specific discovery may be expensive and unneeded while generic I/O discovery could still be useful.

#### 22.11.3.2 int hwloc_topology_set_pid (hwloc_topology_t restrict *topology*, hwloc_pid_t *pid*)

Change which process the topology is viewed from. On some systems, processes may have different views of the machine, for instance the set of allowed CPUs. By default, hwloc exposes the view from the current process. Calling hwloc_topology_set_pid() permits to make it expose the topology of the machine from the point of view of another process.

**Note:**

`hwloc_pid_t` is `pid_t` on Unix platforms, and `HANDLE` on native Windows platforms.
-1 is returned and errno is set to ENOSYS on platforms that do not support this feature.

#### 22.11.3.3 int hwloc_topology_set_synthetic (hwloc_topology_t restrict *topology*, const char *restrict *description*)

Enable synthetic topology. Gather topology information from the given `description`, a space-separated string of <type:number> describing the object type and arity at each level. All types may be omitted (space-separated string of numbers) so that hwloc chooses all types according to usual topologies. See also the Synthetic topologies.

Setting the environment variable HWLOC_SYNTHETIC may also result in this behavior.

If `description` was properly parsed and describes a valid topology configuration, this function returns 0. Otherwise -1 is returned and errno is set to EINVAL.

Note that this function does not actually load topology information; it just tells hwloc where to load it from. You'll still need to invoke hwloc_topology_load() to actually load the topology information.

**Note:**

For convenience, this backend provides empty binding hooks which just return success.
On success, the synthetic component replaces the previously enabled component (if any), but the topology is not actually modified until hwloc_topology_load().

#### 22.11.3.4 int hwloc_topology_set_xml (hwloc_topology_t restrict *topology*, const char *restrict *xmlpath*)

Enable XML-file based topology. Gather topology information from the XML file given at `xmlpath`. Setting the environment variable HWLOC_XMLFILE may also result in this behavior. This file may have been generated earlier with hwloc_topology_export_xml() in hwloc/export.h, or lstopo file.xml.

Note that this function does not actually load topology information; it just tells hwloc where to load it from. You'll still need to invoke hwloc_topology_load() to actually load the topology information.

**Returns:**

-1 with errno set to EINVAL on failure to read the XML file.

**Note:**

See also hwloc_topology_set_userdata_import_callback() for importing application-specific object userdata.
For convenience, this backend provides empty binding hooks which just return success. To have hwloc still actually call OS-specific hooks, the HWLOC_TOPOLOGY_FLAG_IS_THISSYSTEM has to be set to assert that the loaded file is really the underlying system.
On success, the XML component replaces the previously enabled component (if any), but the topology is not actually modified until hwloc_topology_load().

#### 22.11.3.5 int hwloc_topology_set_xmlbuffer (hwloc_topology_t restrict *topology*, const char *restrict *buffer*, int *size*)

Enable XML based topology using a memory buffer (instead of a file, as with hwloc_topology_set_xml()). Gather topology information from the XML memory buffer given at `buffer` and of length `size`. This buffer may have been filled earlier with hwloc_topology_export_xmlbuffer() in hwloc/export.h.

Note that this function does not actually load topology information; it just tells hwloc where to load it from. You'll still need to invoke hwloc_topology_load() to actually load the topology information.

**Returns:**

-1 with errno set to EINVAL on failure to read the XML buffer.

**Note:**

See also hwloc_topology_set_userdata_import_callback() for importing application-specific object userdata.
For convenience, this backend provides empty binding hooks which just return success. To have hwloc still actually call OS-specific hooks, the HWLOC_TOPOLOGY_FLAG_IS_THISSYSTEM has to be set to assert that the loaded file is really the underlying system.
On success, the XML component replaces the previously enabled component (if any), but the topology is not actually modified until hwloc_topology_load().

# 22.12 Topology Detection Configuration and Query

## Data Structures

- struct hwloc_topology_discovery_support

  *Flags describing actual discovery support for this topology.*

- struct hwloc_topology_cpubind_support

  *Flags describing actual PU binding support for this topology.*

- struct hwloc_topology_membind_support

  *Flags describing actual memory binding support for this topology.*

- struct hwloc_topology_support

  *Set of flags describing actual support for this topology.*

## Enumerations

- enum hwloc_topology_flags_e { HWLOC_TOPOLOGY_FLAG_INCLUDE_DISALLOWED, HWLOC_TOPOLOGY_FLAG_IS_THISSYSTEM, HWLOC_TOPOLOGY_FLAG_THISSYSTEM_ALLOWED_RESOURCES }
- enum hwloc_type_filter_e { HWLOC_TYPE_FILTER_KEEP_ALL, HWLOC_TYPE_FILTER_KEEP_NONE, HWLOC_TYPE_FILTER_KEEP_STRUCTURE, HWLOC_TYPE_FILTER_KEEP_IMPORTANT }

## Functions

- int hwloc_topology_set_flags (hwloc_topology_t topology, unsigned long flags)
- unsigned long hwloc_topology_get_flags (hwloc_topology_t topology)
- int hwloc_topology_is_thissystem (hwloc_topology_t restrict topology)
- struct hwloc_topology_support * hwloc_topology_get_support (hwloc_topology_t restrict topology)
- int hwloc_topology_set_type_filter (hwloc_topology_t topology, hwloc_obj_type_t type, enum hwloc_type_filter_e filter)
- int hwloc_topology_get_type_filter (hwloc_topology_t topology, hwloc_obj_type_t type, enum hwloc_type_filter_e *filter)
- int hwloc_topology_set_all_types_filter (hwloc_topology_t topology, enum hwloc_type_filter_e filter)
- int hwloc_topology_set_cache_types_filter (hwloc_topology_t topology, enum hwloc_type_filter_e filter)
- int hwloc_topology_set_icache_types_filter (hwloc_topology_t topology, enum hwloc_type_filter_e filter)
- int hwloc_topology_set_io_types_filter (hwloc_topology_t topology, enum hwloc_type_filter_e filter)
- void hwloc_topology_set_userdata (hwloc_topology_t topology, const void *userdata)
- void * hwloc_topology_get_userdata (hwloc_topology_t topology)

### 22.12.1 Detailed Description

Several functions can optionally be called between hwloc_topology_init() and hwloc_topology_load() to configure how the detection should be performed, e.g. to ignore some objects types, define a synthetic topology, etc.

### 22.12.2 Enumeration Type Documentation

#### 22.12.2.1 enum hwloc_topology_flags_e

Flags to be set onto a topology context before load. Flags should be given to hwloc_topology_set_flags(). They may also be returned by hwloc_topology_get_flags().

**Enumerator:**

***HWLOC_TOPOLOGY_FLAG_INCLUDE_DISALLOWED*** Detect the whole system, ignore reservations, include disallowed objects. Gather all resources, even if some were disabled by the administrator. For instance, ignore Linux Cgroup/Cpusets and gather all processors and memory nodes.

When this flag is not set, PUs and NUMA nodes that are disallowed are not added to the topology. Parent objects (package, core, cache, etc.) are added only if some of their children are allowed. All existing PUs and NUMA nodes in the topology are allowed. hwloc_topology_get_allowed_cpuset() and hwloc_topology_get_allowed_nodeset() are equal to the root object cpuset and nodeset.

When this flag is set, the actual sets of allowed PUs and NUMA nodes are given by hwloc_topology_get_allowed_cpuset() and hwloc_topology_get_allowed_nodeset(). They may be smaller than the root object cpuset and nodeset.

If the current topology is exported to XML and reimported later, this flag should be set again in the reimported topology so that disallowed resources are reimported as well.

***HWLOC_TOPOLOGY_FLAG_IS_THISSYSTEM*** Assume that the selected backend provides the topology for the system on which we are running. This forces hwloc_topology_is_thissystem() to return 1, i.e. makes hwloc assume that the selected backend provides the topology for the system on which we are running, even if it is not the OS-specific backend but the XML backend for instance. This means making the binding functions actually call the OS-specific system calls and really do binding, while the XML backend would otherwise provide empty hooks just returning success.

Setting the environment variable HWLOC_THISSYSTEM may also result in the same behavior.

This can be used for efficiency reasons to first detect the topology once, save it to an XML file, and quickly reload it later through the XML backend, but still having binding functions actually do bind.

***HWLOC_TOPOLOGY_FLAG_THISSYSTEM_ALLOWED_RESOURCES*** Get the set of allowed resources from the local operating system even if the topology was loaded from XML or synthetic description. If the topology was loaded from XML or from a synthetic string, restrict it by applying the current process restrictions such as Linux Cgroup/Cpuset.

This is useful when the topology is not loaded directly from the local machine (e.g. for performance reason) and it comes with all resources, while the running process is restricted to only parts of the machine.

This flag is ignored unless HWLOC_TOPOLOGY_FLAG_IS_THISSYSTEM is also set since the loaded topology must match the underlying machine where restrictions will be gathered from.

Setting the environment variable HWLOC_THISSYSTEM_ALLOWED_RESOURCES would result in the same behavior.

#### 22.12.2.2 enum hwloc_type_filter_e

Type filtering flags. By default, most objects are kept (HWLOC_TYPE_FILTER_KEEP_ALL). Instruction caches, I/O and Misc objects are ignored by default (HWLOC_TYPE_FILTER_KEEP_NONE). Die and Group levels are ignored unless they bring structure (HWLOC_TYPE_FILTER_KEEP_STRUCTURE).

Note that group objects are also ignored individually (without the entire level) when they do not bring structure.

**Enumerator:**

***HWLOC_TYPE_FILTER_KEEP_ALL*** Keep all objects of this type. Cannot be set for HWLOC_OBJ_GROUP (groups are designed only to add more structure to the topology).

***HWLOC_TYPE_FILTER_KEEP_NONE*** Ignore all objects of this type. The bottom-level type HWLOC_OBJ_PU, the HWLOC_OBJ_NUMANODE type, and the top-level type HWLOC_OBJ_MACHINE may not be ignored.

***HWLOC_TYPE_FILTER_KEEP_STRUCTURE*** Only ignore objects if their entire level does not bring any structure. Keep the entire level of objects if at least one of these objects adds structure to the topology. An object brings structure when it has multiple children and it is not the only child of its parent.

If all objects in the level are the only child of their parent, and if none of them has multiple children, the entire level is removed.

Cannot be set for I/O and Misc objects since the topology structure does not matter there.

***HWLOC_TYPE_FILTER_KEEP_IMPORTANT*** Only keep likely-important objects of the given type. It is only useful for I/O object types. For HWLOC_OBJ_PCI_DEVICE and HWLOC_OBJ_OS_DEVICE, it means that only objects of major/common kinds are kept (storage, network, OpenFabrics, Intel MICs, CUDA, OpenCL, NVML, and displays). Also, only OS devices directly attached on PCI (e.g. no USB) are reported. For HWLOC_OBJ_BRIDGE, it means that bridges are kept only if they have children.

This flag equivalent to HWLOC_TYPE_FILTER_KEEP_ALL for Normal, Memory and Misc types since they are likely important.

### 22.12.3 Function Documentation

#### 22.12.3.1 unsigned long hwloc_topology_get_flags (hwloc_topology_t *topology*)

Get OR'ed flags of a topology. Get the OR'ed set of hwloc_topology_flags_e of a topology.

**Returns:**

the flags previously set with hwloc_topology_set_flags().

#### 22.12.3.2 struct hwloc_topology_support* hwloc_topology_get_support (hwloc_topology_t restrict *topology*) `[read]`

Retrieve the topology support. Each flag indicates whether a feature is supported. If set to 0, the feature is not supported. If set to 1, the feature is supported, but the corresponding call may still fail in some corner cases.

These features are also listed by hwloc-info \--support

#### 22.12.3.3 int hwloc_topology_get_type_filter (hwloc_topology_t *topology*, hwloc_obj_type_t *type*, enum hwloc_type_filter_e * *filter*)

Get the current filtering for the given object type.

#### 22.12.3.4 void* hwloc_topology_get_userdata (hwloc_topology_t *topology*)

Retrieve the topology-specific userdata pointer. Retrieve the application-given private data pointer that was previously set with hwloc_topology_set_userdata().

#### 22.12.3.5 int hwloc_topology_is_thissystem (hwloc_topology_t restrict *topology*)

Does the topology context come from this system?

**Returns:**

1 if this topology context was built using the system running this program.
0 instead (for instance if using another file-system root, a XML topology file, or a synthetic topology).

#### 22.12.3.6 int hwloc_topology_set_all_types_filter (hwloc_topology_t *topology*, enum hwloc_type_filter_e *filter*)

Set the filtering for all object types. If some types do not support this filtering, they are silently ignored.

#### 22.12.3.7 int hwloc_topology_set_cache_types_filter (hwloc_topology_t *topology*, enum hwloc_type_filter_e *filter*)

Set the filtering for all CPU cache object types. Memory-side caches are not involved since they are not CPU caches.

#### 22.12.3.8 int hwloc_topology_set_flags (hwloc_topology_t *topology*, unsigned long *flags*)

Set OR'ed flags to non-yet-loaded topology. Set a OR'ed set of hwloc_topology_flags_e onto a topology that was not yet loaded.

If this function is called multiple times, the last invokation will erase and replace the set of flags that was previously set.

The flags set in a topology may be retrieved with hwloc_topology_get_flags()

#### 22.12.3.9 int hwloc_topology_set_icache_types_filter (hwloc_topology_t *topology*, enum hwloc_type_filter_e *filter*)

Set the filtering for all CPU instruction cache object types. Memory-side caches are not involved since they are not CPU caches.

#### 22.12.3.10 int hwloc_topology_set_io_types_filter (hwloc_topology_t *topology*, enum hwloc_type_filter_e *filter*)

Set the filtering for all I/O object types.

#### 22.12.3.11 int hwloc_topology_set_type_filter (hwloc_topology_t *topology*, hwloc_obj_type_t *type*, enum hwloc_type_filter_e *filter*)

Set the filtering for the given object type.

#### 22.12.3.12 void hwloc_topology_set_userdata (hwloc_topology_t *topology*, const void * *userdata*)

Set the topology-specific userdata pointer. Each topology may store one application-given private data pointer. It is initialized to `NULL`. hwloc will never modify it.

Use it as you wish, after hwloc_topology_init() and until hwloc_topolog_destroy().

This pointer is not exported to XML.

## 22.13 Modifying a loaded Topology

### Enumerations

- enum hwloc_restrict_flags_e {
  HWLOC_RESTRICT_FLAG_REMOVE_CPULESS, HWLOC_RESTRICT_FLAG_BYNODESET = (1UL<<3), HWLOC_RESTRICT_FLAG_REMOVE_MEMLESS, HWLOC_RESTRICT_FLAG_ADAPT_MISC,
  HWLOC_RESTRICT_FLAG_ADAPT_IO }
- enum hwloc_allow_flags_e { HWLOC_ALLOW_FLAG_ALL, HWLOC_ALLOW_FLAG_LOCAL_RESTRICTIONS, HWLOC_ALLOW_FLAG_CUSTOM }

### Functions

- int hwloc_topology_restrict (hwloc_topology_t restrict topology, hwloc_const_bitmap_t set, unsigned long flags)
- int hwloc_topology_allow (hwloc_topology_t restrict topology, hwloc_const_cpuset_t cpuset, hwloc_const_nodeset_t nodeset, unsigned long flags)
- hwloc_obj_t hwloc_topology_insert_misc_object (hwloc_topology_t topology, hwloc_obj_t parent, const char *name)
- hwloc_obj_t hwloc_topology_alloc_group_object (hwloc_topology_t topology)
- hwloc_obj_t hwloc_topology_insert_group_object (hwloc_topology_t topology, hwloc_obj_t group)
- int hwloc_obj_add_other_obj_sets (hwloc_obj_t dst, hwloc_obj_t src)

### 22.13.1 Enumeration Type Documentation

#### 22.13.1.1 enum hwloc_allow_flags_e

Flags to be given to hwloc_topology_allow().

**Enumerator:**

***HWLOC_ALLOW_FLAG_ALL*** Mark all objects as allowed in the topology. `cpuset` and `nodeset` given to hwloc_topology_allow() must be `NULL`.

***HWLOC_ALLOW_FLAG_LOCAL_RESTRICTIONS*** Only allow objects that are available to the current process. The topology must have HWLOC_TOPOLOGY_FLAG_IS_THISSYSTEM so that the set of available resources can actually be retrieved from the operating system.
`cpuset` and `nodeset` given to hwloc_topology_allow() must be `NULL`.

***HWLOC_ALLOW_FLAG_CUSTOM*** Allow a custom set of objects, given to hwloc_topology_allow() as `cpuset` and/or `nodeset` parameters.

#### 22.13.1.2 enum hwloc_restrict_flags_e

Flags to be given to hwloc_topology_restrict().

**Enumerator:**

***HWLOC_RESTRICT_FLAG_REMOVE_CPULESS*** Remove all objects that became CPU-less. By default, only objects that contain no PU and no memory are removed. This flag may not be used with HWLOC_RESTRICT_FLAG_BYNODESET.

***HWLOC_RESTRICT_FLAG_BYNODESET*** Restrict by nodeset instead of CPU set. Only keep objects whose nodeset is included or partially included in the given set. This flag may not be used with HWLOC_RESTRICT_FLAG_REMOVE_CPULESS.

***HWLOC_RESTRICT_FLAG_REMOVE_MEMLESS*** Remove all objects that became Memoryless. By default, only objects that contain no PU and no memory are removed. This flag may only be used with HWLOC_RESTRICT_FLAG_BYNODESET.

***HWLOC_RESTRICT_FLAG_ADAPT_MISC*** Move Misc objects to ancestors if their parents are removed during restriction. If this flag is not set, Misc objects are removed when their parents are removed.

***HWLOC_RESTRICT_FLAG_ADAPT_IO*** Move I/O objects to ancestors if their parents are removed during restriction. If this flag is not set, I/O devices and bridges are removed when their parents are removed.

## 22.13.2 Function Documentation

### 22.13.2.1 int hwloc_obj_add_other_obj_sets (hwloc_obj_t *dst*, hwloc_obj_t *src*)

Setup object cpusets/nodesets by OR'ing another object's sets. For each defined cpuset or nodeset in `src`, allocate the corresponding set in `dst` and add `src` to it by OR'ing sets.

This function is convenient between hwloc_topology_alloc_group_object() and hwloc_topology_insert_group_object(). It builds the sets of the new Group that will be inserted as a new intermediate parent of several objects.

### 22.13.2.2 hwloc_obj_t hwloc_topology_alloc_group_object (hwloc_topology_t *topology*)

Allocate a Group object to insert later with hwloc_topology_insert_group_object(). This function returns a new Group object. The caller should (at least) initialize its sets before inserting the object. See hwloc_topology_insert_group_object().

The `subtype` object attribute may be set to display something else than "Group" as the type name for this object in lstopo. Custom name/value info pairs may be added with hwloc_obj_add_info() after insertion.

The `kind` group attribute should be 0. The `subkind` group attribute may be set to identify multiple Groups of the same level.

It is recommended not to set any other object attribute before insertion, since the Group may get discarded during insertion.

The object will be destroyed if passed to hwloc_topology_insert_group_object() without any set defined.

### 22.13.2.3 int hwloc_topology_allow (hwloc_topology_t restrict *topology*, hwloc_const_cpuset_t *cpuset*, hwloc_const_nodeset_t *nodeset*, unsigned long *flags*)

Change the sets of allowed PUs and NUMA nodes in the topology. This function only works if the HWLOC_TOPOLOGY_FLAG_INCLUDE_DISALLOWED was set on the topology. It does not modify any object, it only changes the sets returned by hwloc_topology_get_allowed_cpuset() and hwloc_topology_get_allowed_nodeset().

It is notably useful when importing a topology from another process running in a different Linux Cgroup.

`flags` must be set to one flag among hwloc_allow_flags_e.

**Note:**

Removing objects from a topology should rather be performed with hwloc_topology_restrict().

#### 22.13.2.4 hwloc_obj_t hwloc_topology_insert_group_object (hwloc_topology_t *topology*, hwloc_obj_t *group*)

Add more structure to the topology by adding an intermediate Group. The caller should first allocate a new Group object with hwloc_topology_alloc_group_object(). Then it must setup at least one of its CPU or node sets to specify the final location of the Group in the topology. Then the object can be passed to this function for actual insertion in the topology.

The group `dont_merge` attribute may be set to prevent the core from ever merging this object with another object hierarchically-identical.

Either the cpuset or nodeset field (or both, if compatible) must be set to a non-empty bitmap. The complete_cpuset or complete_nodeset may be set instead if inserting with respect to the complete topology (including disallowed, offline or unknown objects).

It grouping several objects, hwloc_obj_add_other_obj_sets() is an easy way to build the Group sets iteratively.

These sets cannot be larger than the current topology, or they would get restricted silently.

The core will setup the other sets after actual insertion.

**Returns:**

The inserted object if it was properly inserted.
An existing object if the Group was discarded because the topology already contained an object at the same location (the Group did not add any locality information). Any name/info key pair set before inserting is appended to the existing object.
`NULL` if the insertion failed because of conflicting sets in topology tree.
`NULL` if Group objects are filtered-out of the topology (HWLOC_TYPE_FILTER_KEEP_NONE).
`NULL` if the object was discarded because no set was initialized in the Group before insert, or all of them were empty.

#### 22.13.2.5 hwloc_obj_t hwloc_topology_insert_misc_object (hwloc_topology_t *topology*, hwloc_obj_t *parent*, const char * *name*)

Add a MISC object as a leaf of the topology. A new MISC object will be created and inserted into the topology at the position given by parent. It is appended to the list of existing Misc children, without ever adding any intermediate hierarchy level. This is useful for annotating the topology without actually changing the hierarchy.

`name` is supposed to be unique across all Misc objects in the topology. It will be duplicated to setup the new object attributes.

The new leaf object will not have any `cpuset`.

**Returns:**

the newly-created object
`NULL` on error.
`NULL` if Misc objects are filtered-out of the topology (HWLOC_TYPE_FILTER_KEEP_NONE).

**Note:**

If `name` contains some non-printable characters, they will be dropped when exporting to XML, see hwloc_topology_export_xml() in hwloc/export.h.

#### 22.13.2.6 int hwloc_topology_restrict (hwloc_topology_t restrict *topology*, hwloc_const_bitmap_t *set*, unsigned long *flags*)

Restrict the topology to the given CPU set or nodeset. Topology `topology` is modified so as to remove all objects that are not included (or partially included) in the CPU set `set`. All objects CPU and node sets are restricted accordingly.

If HWLOC_RESTRICT_FLAG_BYNODESET is passed in `flags`, `set` is considered a nodeset instead of a CPU set.

`flags` is a OR'ed set of hwloc_restrict_flags_e.

**Note:**

This call may not be reverted by restricting back to a larger set. Once dropped during restriction, objects may not be brought back, except by loading another topology with hwloc_topology_load().

**Returns:**

0 on success.
-1 with errno set to EINVAL if the input set is invalid. The topology is not modified in this case.
-1 with errno set to ENOMEM on failure to allocate internal data. The topology is reinitialized in this case. It should be either destroyed with hwloc_topology_destroy() or configured and loaded again.

## 22.14 Finding Objects inside a CPU set

### Functions

- static hwloc_obj_t hwloc_get_first_largest_obj_inside_cpuset (hwloc_topology_t topology, hwloc_const_cpuset_t set)
- int hwloc_get_largest_objs_inside_cpuset (hwloc_topology_t topology, hwloc_const_cpuset_t set, hwloc_obj_t *restrict objs, int max)
- static hwloc_obj_t hwloc_get_next_obj_inside_cpuset_by_depth (hwloc_topology_t topology, hwloc_const_cpuset_t set, int depth, hwloc_obj_t prev)
- static hwloc_obj_t hwloc_get_next_obj_inside_cpuset_by_type (hwloc_topology_t topology, hwloc_const_cpuset_t set, hwloc_obj_type_t type, hwloc_obj_t prev)
- static hwloc_obj_t hwloc_get_obj_inside_cpuset_by_depth (hwloc_topology_t topology, hwloc_const_cpuset_t set, int depth, unsigned idx)
- static hwloc_obj_t hwloc_get_obj_inside_cpuset_by_type (hwloc_topology_t topology, hwloc_const_cpuset_t set, hwloc_obj_type_t type, unsigned idx)
- static unsigned hwloc_get_nbobjs_inside_cpuset_by_depth (hwloc_topology_t topology, hwloc_const_cpuset_t set, int depth)
- static int hwloc_get_nbobjs_inside_cpuset_by_type (hwloc_topology_t topology, hwloc_const_cpuset_t set, hwloc_obj_type_t type)
- static int hwloc_get_obj_index_inside_cpuset (hwloc_topology_t topology, hwloc_const_cpuset_t set, hwloc_obj_t obj)

### 22.14.1 Function Documentation

#### 22.14.1.1 static hwloc_obj_t hwloc_get_first_largest_obj_inside_cpuset (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*) [inline, static]

Get the first largest object included in the given cpuset `set`.

**Returns:**

the first object that is included in `set` and whose parent is not.

This is convenient for iterating over all largest objects within a CPU set by doing a loop getting the first largest object and clearing its CPU set from the remaining CPU set.

#### 22.14.1.2 int hwloc_get_largest_objs_inside_cpuset (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*, hwloc_obj_t *restrict *objs*, int *max*)

Get the set of largest objects covering exactly a given cpuset `set`.

**Returns:**

the number of objects returned in `objs`.

#### 22.14.1.3 static unsigned hwloc_get_nbobjs_inside_cpuset_by_depth (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*, int *depth*) [inline, static]

Return the number of objects at depth `depth` included in CPU set `set`.

**Note:**

Objects with empty CPU sets are ignored (otherwise they would be considered included in any given set).
This function cannot work if objects at the given depth do not have CPU sets (I/O or Misc objects).

#### 22.14.1.4 static int hwloc_get_nbobjs_inside_cpuset_by_type (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*, hwloc_obj_type_t *type*) [inline, static]

Return the number of objects of type `type` included in CPU set `set`. If no object for that type exists inside CPU set `set`, 0 is returned. If there are several levels with objects of that type inside CPU set `set`, -1 is returned.

**Note:**

Objects with empty CPU sets are ignored (otherwise they would be considered included in any given set).
This function cannot work if objects of the given type do not have CPU sets (I/O objects).

#### 22.14.1.5 static hwloc_obj_t hwloc_get_next_obj_inside_cpuset_by_depth (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*, int *depth*, hwloc_obj_t *prev*) [inline, static]

Return the next object at depth `depth` included in CPU set `set`. If `prev` is `NULL`, return the first object at depth `depth` included in `set`. The next invokation should pass the previous return value in `prev` so as to obtain the next object in `set`.

**Note:**

Objects with empty CPU sets are ignored (otherwise they would be considered included in any given set).
This function cannot work if objects at the given depth do not have CPU sets (I/O or Misc objects).

#### 22.14.1.6 static hwloc_obj_t hwloc_get_next_obj_inside_cpuset_by_type (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*, hwloc_obj_type_t *type*, hwloc_obj_t *prev*) [inline, static]

Return the next object of type `type` included in CPU set `set`. If there are multiple or no depth for given type, return `NULL` and let the caller fallback to hwloc_get_next_obj_inside_cpuset_by_depth().

**Note:**

Objects with empty CPU sets are ignored (otherwise they would be considered included in any given set).
This function cannot work if objects of the given type do not have CPU sets (I/O or Misc objects).

#### 22.14.1.7 static int hwloc_get_obj_index_inside_cpuset (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*, hwloc_obj_t *obj*) [inline, static]

Return the logical index among the objects included in CPU set `set`. Consult all objects in the same level as `obj` and inside CPU set `set` in the logical order, and return the index of `obj` within them. If `set`

covers the entire topology, this is the logical index of `obj`. Otherwise, this is similar to a logical index within the part of the topology defined by CPU set `set`.

**Note:**

Objects with empty CPU sets are ignored (otherwise they would be considered included in any given set).
This function cannot work if obj does not have CPU sets (I/O objects).

#### 22.14.1.8 static hwloc_obj_t hwloc_get_obj_inside_cpuset_by_depth (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*, int *depth*, unsigned *idx*) [inline, static]

Return the (logically) `idx` -th object at depth `depth` included in CPU set `set`.

**Note:**

Objects with empty CPU sets are ignored (otherwise they would be considered included in any given set).
This function cannot work if objects at the given depth do not have CPU sets (I/O or Misc objects).

#### 22.14.1.9 static hwloc_obj_t hwloc_get_obj_inside_cpuset_by_type (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*, hwloc_obj_type_t *type*, unsigned *idx*) [inline, static]

Return the `idx` -th object of type `type` included in CPU set `set`. If there are multiple or no depth for given type, return `NULL` and let the caller fallback to hwloc_get_obj_inside_cpuset_by_depth().

**Note:**

Objects with empty CPU sets are ignored (otherwise they would be considered included in any given set).
This function cannot work if objects of the given type do not have CPU sets (I/O or Misc objects).

# 22.15 Finding Objects covering at least CPU set

## Functions

- static hwloc_obj_t hwloc_get_child_covering_cpuset (hwloc_topology_t topology, hwloc_const_cpuset_t set, hwloc_obj_t parent)
- static hwloc_obj_t hwloc_get_obj_covering_cpuset (hwloc_topology_t topology, hwloc_const_cpuset_t set)
- static hwloc_obj_t hwloc_get_next_obj_covering_cpuset_by_depth (hwloc_topology_t topology, hwloc_const_cpuset_t set, int depth, hwloc_obj_t prev)
- static hwloc_obj_t hwloc_get_next_obj_covering_cpuset_by_type (hwloc_topology_t topology, hwloc_const_cpuset_t set, hwloc_obj_type_t type, hwloc_obj_t prev)

## 22.15.1 Function Documentation

### 22.15.1.1 static hwloc_obj_t hwloc_get_child_covering_cpuset (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*, hwloc_obj_t *parent*) [inline, static]

Get the child covering at least CPU set `set`.

**Returns:**

`NULL` if no child matches or if `set` is empty.

**Note:**

This function cannot work if parent does not have a CPU set (I/O or Misc objects).

### 22.15.1.2 static hwloc_obj_t hwloc_get_next_obj_covering_cpuset_by_depth (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*, int *depth*, hwloc_obj_t *prev*) [inline, static]

Iterate through same-depth objects covering at least CPU set `set`. If object `prev` is `NULL`, return the first object at depth `depth` covering at least part of CPU set `set`. The next invokation should pass the previous return value in `prev` so as to obtain the next object covering at least another part of `set`.

**Note:**

This function cannot work if objects at the given depth do not have CPU sets (I/O or Misc objects).

### 22.15.1.3 static hwloc_obj_t hwloc_get_next_obj_covering_cpuset_by_type (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*, hwloc_obj_type_t *type*, hwloc_obj_t *prev*) [inline, static]

Iterate through same-type objects covering at least CPU set `set`. If object `prev` is `NULL`, return the first object of type `type` covering at least part of CPU set `set`. The next invokation should pass the previous return value in `prev` so as to obtain the next object of type `type` covering at least another part of `set`.

If there are no or multiple depths for type `type`, `NULL` is returned. The caller may fallback to hwloc_get_next_obj_covering_cpuset_by_depth() for each depth.

**Note:**

This function cannot work if objects of the given type do not have CPU sets (I/O or Misc objects).

#### 22.15.1.4 static hwloc_obj_t hwloc_get_obj_covering_cpuset (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*) [inline, static]

Get the lowest object covering at least CPU set `set`.

**Returns:**

`NULL` if no object matches or if `set` is empty.

# 22.16 Looking at Ancestor and Child Objects

## Functions

- static hwloc_obj_t hwloc_get_ancestor_obj_by_depth (hwloc_topology_t topology, int depth, hwloc_obj_t obj)
- static hwloc_obj_t hwloc_get_ancestor_obj_by_type (hwloc_topology_t topology, hwloc_obj_type_t type, hwloc_obj_t obj)
- static hwloc_obj_t hwloc_get_common_ancestor_obj (hwloc_topology_t topology, hwloc_obj_t obj1, hwloc_obj_t obj2)
- static int hwloc_obj_is_in_subtree (hwloc_topology_t topology, hwloc_obj_t obj, hwloc_obj_t subtree_root)
- static hwloc_obj_t hwloc_get_next_child (hwloc_topology_t topology, hwloc_obj_t parent, hwloc_obj_t prev)

## 22.16.1 Detailed Description

Be sure to see the figure in Terms and Definitions that shows a complete topology tree, including depths, child/sibling/cousin relationships, and an example of an asymmetric topology where one package has fewer caches than its peers.

## 22.16.2 Function Documentation

### 22.16.2.1 static hwloc_obj_t hwloc_get_ancestor_obj_by_depth (hwloc_topology_t *topology*, int *depth*, hwloc_obj_t *obj*) [inline, static]

Returns the ancestor object of `obj` at depth `depth`.

**Note:**

`depth` should not be the depth of PU or NUMA objects since they are ancestors of no objects (except Misc or I/O). This function rather expects an intermediate level depth, such as the depth of Packages, Cores, or Caches.

### 22.16.2.2 static hwloc_obj_t hwloc_get_ancestor_obj_by_type (hwloc_topology_t *topology*, hwloc_obj_type_t *type*, hwloc_obj_t *obj*) [inline, static]

Returns the ancestor object of `obj` with type `type`.

**Note:**

`type` should not be HWLOC_OBJ_PU or HWLOC_OBJ_NUMANODE since these objects are ancestors of no objects (except Misc or I/O). This function rather expects an intermediate object type, such as HWLOC_OBJ_PACKAGE, HWLOC_OBJ_CORE, etc.

### 22.16.2.3 static hwloc_obj_t hwloc_get_common_ancestor_obj (hwloc_topology_t *topology*, hwloc_obj_t *obj1*, hwloc_obj_t *obj2*) [inline, static]

Returns the common parent object to objects `obj1` and `obj2`.

#### 22.16.2.4 static hwloc_obj_t hwloc_get_next_child (hwloc_topology_t *topology*, hwloc_obj_t *parent*, hwloc_obj_t *prev*) [inline, static]

Return the next child. Return the next child among the normal children list, then among the memory children list, then among the I/O children list, then among the Misc children list.

If `prev` is `NULL`, return the first child.

Return `NULL` when there is no next child.

#### 22.16.2.5 static int hwloc_obj_is_in_subtree (hwloc_topology_t *topology*, hwloc_obj_t *obj*, hwloc_obj_t *subtree_root*) [inline, static]

Returns true if `obj` is inside the subtree beginning with ancestor object `subtree_root`.

**Note:**

This function cannot work if `obj` and `subtree_root` objects do not have CPU sets (I/O or Misc objects).

# 22.17 Kinds of object Type

## Functions

- int hwloc_obj_type_is_normal (hwloc_obj_type_t type)
- int hwloc_obj_type_is_io (hwloc_obj_type_t type)
- int hwloc_obj_type_is_memory (hwloc_obj_type_t type)
- int hwloc_obj_type_is_cache (hwloc_obj_type_t type)
- int hwloc_obj_type_is_dcache (hwloc_obj_type_t type)
- int hwloc_obj_type_is_icache (hwloc_obj_type_t type)

## 22.17.1 Detailed Description

Each object type is either Normal (i.e. hwloc_obj_type_is_normal() returns 1), or Memory (i.e. hwloc_obj_type_is_memory() returns 1) or I/O (i.e. hwloc_obj_type_is_io() returns 1) or Misc (i.e. equal to HWLOC_OBJ_MISC). It cannot be of more than one of these kinds.

## 22.17.2 Function Documentation

### 22.17.2.1 int hwloc_obj_type_is_cache (hwloc_obj_type_t *type*)

Check whether an object type is a CPU Cache (Data, Unified or Instruction). Memory-side caches are not CPU caches.

**Returns:**

1 if an object of type `type` is a Cache, 0 otherwise.

### 22.17.2.2 int hwloc_obj_type_is_dcache (hwloc_obj_type_t *type*)

Check whether an object type is a CPU Data or Unified Cache. Memory-side caches are not CPU caches.

**Returns:**

1 if an object of type `type` is a CPU Data or Unified Cache, 0 otherwise.

### 22.17.2.3 int hwloc_obj_type_is_icache (hwloc_obj_type_t *type*)

Check whether an object type is a CPU Instruction Cache,. Memory-side caches are not CPU caches.

**Returns:**

1 if an object of type `type` is a CPU Instruction Cache, 0 otherwise.

#### 22.17.2.4 int hwloc_obj_type_is_io (hwloc_obj_type_t *type*)

Check whether an object type is I/O. I/O objects are objects attached to their parents in the I/O children list. This current includes Bridges, PCI and OS devices.

**Returns:**

1 if an object of type `type` is a I/O object, 0 otherwise.

#### 22.17.2.5 int hwloc_obj_type_is_memory (hwloc_obj_type_t *type*)

Check whether an object type is Memory. Memory objects are objects attached to their parents in the Memory children list. This current includes NUMA nodes and Memory-side caches.

**Returns:**

1 if an object of type `type` is a Memory object, 0 otherwise.

#### 22.17.2.6 int hwloc_obj_type_is_normal (hwloc_obj_type_t *type*)

Check whether an object type is Normal. Normal objects are objects of the main CPU hierarchy (Machine, Package, Core, PU, CPU caches, etc.), but they are not NUMA nodes, I/O devices or Misc objects.

They are attached to parent as Normal children, not as Memory, I/O or Misc children.

**Returns:**

1 if an object of type `type` is a Normal object, 0 otherwise.

# 22.18 Looking at Cache Objects

## Functions

- static int hwloc_get_cache_type_depth (hwloc_topology_t topology, unsigned cachelevel, hwloc_obj_cache_type_t cachetype)
- static hwloc_obj_t hwloc_get_cache_covering_cpuset (hwloc_topology_t topology, hwloc_const_cpuset_t set)
- static hwloc_obj_t hwloc_get_shared_cache_covering_obj (hwloc_topology_t topology, hwloc_obj_t obj)

## 22.18.1 Function Documentation

### 22.18.1.1 static hwloc_obj_t hwloc_get_cache_covering_cpuset (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*) [inline, static]

Get the first data (or unified) cache covering a cpuset `set`.

**Returns:**

`NULL` if no cache matches.

### 22.18.1.2 static int hwloc_get_cache_type_depth (hwloc_topology_t *topology*, unsigned *cachelevel*, hwloc_obj_cache_type_t *cachetype*) [inline, static]

Find the depth of cache objects matching cache level and type. Return the depth of the topology level that contains cache objects whose attributes match `cachelevel` and `cachetype`.

This function is identical to calling hwloc_get_type_depth() with the corresponding type such as HWLOC_OBJ_L1ICACHE, except that it may also return a Unified cache when looking for an instruction cache.

If no cache level matches, HWLOC_TYPE_DEPTH_UNKNOWN is returned.

If `cachetype` is HWLOC_OBJ_CACHE_UNIFIED, the depth of the unique matching unified cache level is returned.

If `cachetype` is HWLOC_OBJ_CACHE_DATA or HWLOC_OBJ_CACHE_INSTRUCTION, either a matching cache, or a unified cache is returned.

If `cachetype` is `-1`, it is ignored and multiple levels may match. The function returns either the depth of a uniquely matching level or HWLOC_TYPE_DEPTH_MULTIPLE.

### 22.18.1.3 static hwloc_obj_t hwloc_get_shared_cache_covering_obj (hwloc_topology_t *topology*, hwloc_obj_t *obj*) [inline, static]

Get the first data (or unified) cache shared between an object and somebody else.

**Returns:**

`NULL` if no cache matches or if an invalid object is given.

## 22.19 Finding objects, miscellaneous helpers

### Functions

- int hwloc_bitmap_singlify_per_core (hwloc_topology_t topology, hwloc_bitmap_t cpuset, unsigned which)
- static hwloc_obj_t hwloc_get_pu_obj_by_os_index (hwloc_topology_t topology, unsigned os_index)
- static hwloc_obj_t hwloc_get_numanode_obj_by_os_index (hwloc_topology_t topology, unsigned os_index)
- unsigned hwloc_get_closest_objs (hwloc_topology_t topology, hwloc_obj_t src, hwloc_obj_t *restrict objs, unsigned max)
- static hwloc_obj_t hwloc_get_obj_below_by_type (hwloc_topology_t topology, hwloc_obj_type_t type1, unsigned idx1, hwloc_obj_type_t type2, unsigned idx2)
- static hwloc_obj_t hwloc_get_obj_below_array_by_type (hwloc_topology_t topology, int nr, hwloc_obj_type_t *typev, unsigned *idxv)

### 22.19.1 Detailed Description

Be sure to see the figure in Terms and Definitions that shows a complete topology tree, including depths, child/sibling/cousin relationships, and an example of an asymmetric topology where one package has fewer caches than its peers.

### 22.19.2 Function Documentation

#### 22.19.2.1 int hwloc_bitmap_singlify_per_core (hwloc_topology_t *topology*, hwloc_bitmap_t *cpuset*, unsigned *which*)

Remove simultaneous multithreading PUs from a CPU set. For each core in `topology`, if `cpuset` contains some PUs of that core, modify `cpuset` to only keep a single PU for that core.

`which` specifies which PU will be kept. PU are considered in physical index order. If 0, for each core, the function keeps the first PU that was originally set in `cpuset`.

If `which` is larger than the number of PUs in a core there were originally set in `cpuset`, no PU is kept for that core.

**Note:**

PUs that are not below a Core object are ignored (for instance if the topology does not contain any Core object). None of them is removed from `cpuset`.

#### 22.19.2.2 unsigned hwloc_get_closest_objs (hwloc_topology_t *topology*, hwloc_obj_t *src*, hwloc_obj_t *restrict *objs*, unsigned *max*)

Do a depth-first traversal of the topology to find and sort. all objects that are at the same depth than `src`. Report in `objs` up to `max` physically closest ones to `src`.

**Returns:**

the number of objects returned in `objs`.
0 if `src` is an I/O object.

**Note:**

This function requires the `src` object to have a CPU set.

#### 22.19.2.3 static hwloc_obj_t hwloc_get_numanode_obj_by_os_index (hwloc_topology_t *topology*, unsigned *os_index*) [inline, static]

Returns the object of type HWLOC_OBJ_NUMANODE with `os_index`. This function is useful for converting a nodeset into the NUMA node objects it contains. When retrieving the current binding (e.g. with hwloc_get_membind() with HWLOC_MEMBIND_BYNODESET), one may iterate over the bits of the resulting nodeset with hwloc_bitmap_foreach_begin(), and find the corresponding NUMA nodes with this function.

#### 22.19.2.4 static hwloc_obj_t hwloc_get_obj_below_array_by_type (hwloc_topology_t *topology*, int *nr*, hwloc_obj_type_t * *typev*, unsigned * *idxv*) [inline, static]

Find an object below a chain of objects specified by types and indexes. This is a generalized version of hwloc_get_obj_below_by_type().

Arrays `typev` and `idxv` must contain `nr` types and indexes.

Start from the top system object and walk the arrays `typev` and `idxv`. For each type and logical index couple in the arrays, look under the previously found object to find the index-th object of the given type. Indexes are specified within the parent, not withing the entire system.

For instance, if nr is 3, typev contains NODE, PACKAGE and CORE, and idxv contains 0, 1 and 2, return the third core object below the second package below the first NUMA node.

**Note:**

This function requires all these objects and the root object to have a CPU set.

#### 22.19.2.5 static hwloc_obj_t hwloc_get_obj_below_by_type (hwloc_topology_t *topology*, hwloc_obj_type_t *type1*, unsigned *idx1*, hwloc_obj_type_t *type2*, unsigned *idx2*) [inline, static]

Find an object below another object, both specified by types and indexes. Start from the top system object and find object of type `type1` and logical index `idx1`. Then look below this object and find another object of type `type2` and logical index `idx2`. Indexes are specified within the parent, not withing the entire system.

For instance, if type1 is PACKAGE, idx1 is 2, type2 is CORE and idx2 is 3, return the fourth core object below the third package.

**Note:**

This function requires these objects to have a CPU set.

#### 22.19.2.6 static hwloc_obj_t hwloc_get_pu_obj_by_os_index (hwloc_topology_t *topology*, unsigned *os_index*) [inline, static]

Returns the object of type HWLOC_OBJ_PU with `os_index`. This function is useful for converting a CPU set into the PU objects it contains. When retrieving the current binding (e.g. with hwloc_get_-

cpubind()), one may iterate over the bits of the resulting CPU set with hwloc_bitmap_foreach_begin(), and find the corresponding PUs with this function.

# 22.20 Distributing items over a topology

## Enumerations

- enum hwloc_distrib_flags_e { HWLOC_DISTRIB_FLAG_REVERSE }

## Functions

- static int hwloc_distrib (hwloc_topology_t topology, hwloc_obj_t *roots, unsigned n_roots, hwloc_cpuset_t *set, unsigned n, int until, unsigned long flags)

## 22.20.1 Enumeration Type Documentation

### 22.20.1.1 enum hwloc_distrib_flags_e

Flags to be given to hwloc_distrib().

**Enumerator:**

***HWLOC_DISTRIB_FLAG_REVERSE*** Distrib in reverse order, starting from the last objects.

## 22.20.2 Function Documentation

### 22.20.2.1 static int hwloc_distrib (hwloc_topology_t *topology*, hwloc_obj_t * *roots*, unsigned *n_roots*, hwloc_cpuset_t * *set*, unsigned *n*, int *until*, unsigned long *flags*) [inline, static]

Distribute `n` items over the topology under `roots`. Array `set` will be filled with `n` cpusets recursively distributed linearly over the topology under objects `roots`, down to depth `until` (which can be INT_MAX to distribute down to the finest level).

`n_roots` is usually 1 and `roots` only contains the topology root object so as to distribute over the entire topology.

This is typically useful when an application wants to distribute `n` threads over a machine, giving each of them as much private cache as possible and keeping them locally in number order.

The caller may typically want to also call hwloc_bitmap_singlify() before binding a thread so that it does not move at all.

`flags` should be 0 or a OR'ed set of hwloc_distrib_flags_e.

**Note:**

This function requires the `roots` objects to have a CPU set.
This function replaces the now deprecated hwloc_distribute() and hwloc_distributev() functions.

## 22.21 CPU and node sets of entire topologies

### Functions

- hwloc_const_cpuset_t hwloc_topology_get_complete_cpuset (hwloc_topology_t topology)
- hwloc_const_cpuset_t hwloc_topology_get_topology_cpuset (hwloc_topology_t topology)
- hwloc_const_cpuset_t hwloc_topology_get_allowed_cpuset (hwloc_topology_t topology)
- hwloc_const_nodeset_t hwloc_topology_get_complete_nodeset (hwloc_topology_t topology)
- hwloc_const_nodeset_t hwloc_topology_get_topology_nodeset (hwloc_topology_t topology)
- hwloc_const_nodeset_t hwloc_topology_get_allowed_nodeset (hwloc_topology_t topology)

### 22.21.1 Function Documentation

#### 22.21.1.1 hwloc_const_cpuset_t hwloc_topology_get_allowed_cpuset (hwloc_topology_t *topology*)

Get allowed CPU set.

**Returns:**

the CPU set of allowed logical processors of the system.

**Note:**

If the topology flag HWLOC_TOPOLOGY_FLAG_INCLUDE_DISALLOWED was not set, this is identical to hwloc_topology_get_topology_cpuset(), which means all PUs are allowed.
If HWLOC_TOPOLOGY_FLAG_INCLUDE_DISALLOWED was set, applying hwloc_bitmap_intersects() on the result of this function and on an object cpuset checks whether there are allowed PUs inside that object. Applying hwloc_bitmap_and() returns the list of these allowed PUs.
The returned cpuset is not newly allocated and should thus not be changed or freed, hwloc_bitmap_dup() must be used to obtain a local copy.

#### 22.21.1.2 hwloc_const_nodeset_t hwloc_topology_get_allowed_nodeset (hwloc_topology_t *topology*)

Get allowed node set.

**Returns:**

the node set of allowed memory of the system.

**Note:**

If the topology flag HWLOC_TOPOLOGY_FLAG_INCLUDE_DISALLOWED was not set, this is identical to hwloc_topology_get_topology_nodeset(), which means all NUMA nodes are allowed.
If HWLOC_TOPOLOGY_FLAG_INCLUDE_DISALLOWED was set, applying hwloc_bitmap_intersects() on the result of this function and on an object nodeset checks whether there are allowed NUMA nodes inside that object. Applying hwloc_bitmap_and() returns the list of these allowed NUMA nodes.
The returned nodeset is not newly allocated and should thus not be changed or freed, hwloc_bitmap_dup() must be used to obtain a local copy.

#### 22.21.1.3 hwloc_const_cpuset_t hwloc_topology_get_complete_cpuset (hwloc_topology_t *topology*)

Get complete CPU set.

**Returns:**

the complete CPU set of logical processors of the system.

**Note:**

The returned cpuset is not newly allocated and should thus not be changed or freed; hwloc_bitmap_dup() must be used to obtain a local copy.
This is equivalent to retrieving the root object complete CPU-set.

#### 22.21.1.4 hwloc_const_nodeset_t hwloc_topology_get_complete_nodeset (hwloc_topology_t *topology*)

Get complete node set.

**Returns:**

the complete node set of memory of the system.

**Note:**

The returned nodeset is not newly allocated and should thus not be changed or freed; hwloc_bitmap_dup() must be used to obtain a local copy.
This is equivalent to retrieving the root object complete nodeset.

#### 22.21.1.5 hwloc_const_cpuset_t hwloc_topology_get_topology_cpuset (hwloc_topology_t *topology*)

Get topology CPU set.

**Returns:**

the CPU set of logical processors of the system for which hwloc provides topology information. This is equivalent to the cpuset of the system object.

**Note:**

The returned cpuset is not newly allocated and should thus not be changed or freed; hwloc_bitmap_dup() must be used to obtain a local copy.
This is equivalent to retrieving the root object CPU-set.

#### 22.21.1.6 hwloc_const_nodeset_t hwloc_topology_get_topology_nodeset (hwloc_topology_t *topology*)

Get topology node set.

**Returns:**

the node set of memory of the system for which hwloc provides topology information. This is equivalent to the nodeset of the system object.

**Note:**

The returned nodeset is not newly allocated and should thus not be changed or freed; hwloc_bitmap_dup() must be used to obtain a local copy.
This is equivalent to retrieving the root object nodeset.

# 22.22 Converting between CPU sets and node sets

## Functions

- static int hwloc_cpuset_to_nodeset (hwloc_topology_t topology, hwloc_const_cpuset_t _cpuset, hwloc_nodeset_t nodeset)
- static int hwloc_cpuset_from_nodeset (hwloc_topology_t topology, hwloc_cpuset_t _cpuset, hwloc_const_nodeset_t nodeset)

## 22.22.1 Function Documentation

### 22.22.1.1 static int hwloc_cpuset_from_nodeset (hwloc_topology_t *topology*, hwloc_cpuset_t *_cpuset*, hwloc_const_nodeset_t *nodeset*) [inline, static]

Convert a NUMA node set into a CPU set. For each NUMA node included in the input `nodeset`, set the corresponding local PUs in the output `_cpuset`.

If some CPUs have no local NUMA nodes, this function never sets their indexes in the output CPU set, even if a full node set is given in input.

Hence the entire topology node set is converted into the set of all CPUs that have some local NUMA nodes.

### 22.22.1.2 static int hwloc_cpuset_to_nodeset (hwloc_topology_t *topology*, hwloc_const_cpuset_t *_cpuset*, hwloc_nodeset_t *nodeset*) [inline, static]

Convert a CPU set into a NUMA node set. For each PU included in the input `_cpuset`, set the corresponding local NUMA node(s) in the output `nodeset`.

If some NUMA nodes have no CPUs at all, this function never sets their indexes in the output node set, even if a full CPU set is given in input.

Hence the entire topology CPU set is converted into the set of all nodes that have some local CPUs.

## 22.23 Finding I/O objects

### Functions

- static hwloc_obj_t hwloc_get_non_io_ancestor_obj (hwloc_topology_t topology, hwloc_obj_t ioobj)
- static hwloc_obj_t hwloc_get_next_pcidev (hwloc_topology_t topology, hwloc_obj_t prev)
- static hwloc_obj_t hwloc_get_pcidev_by_busid (hwloc_topology_t topology, unsigned domain, unsigned bus, unsigned dev, unsigned func)
- static hwloc_obj_t hwloc_get_pcidev_by_busidstring (hwloc_topology_t topology, const char *busid)
- static hwloc_obj_t hwloc_get_next_osdev (hwloc_topology_t topology, hwloc_obj_t prev)
- static hwloc_obj_t hwloc_get_next_bridge (hwloc_topology_t topology, hwloc_obj_t prev)
- static int hwloc_bridge_covers_pcibus (hwloc_obj_t bridge, unsigned domain, unsigned bus)

### 22.23.1 Function Documentation

#### 22.23.1.1 static int hwloc_bridge_covers_pcibus (hwloc_obj_t *bridge*, unsigned *domain*, unsigned *bus*) [inline, static]

#### 22.23.1.2 static hwloc_obj_t hwloc_get_next_bridge (hwloc_topology_t *topology*, hwloc_obj_t *prev*) [inline, static]

Get the next bridge in the system.

**Returns:**

the first bridge if `prev` is `NULL`.

#### 22.23.1.3 static hwloc_obj_t hwloc_get_next_osdev (hwloc_topology_t *topology*, hwloc_obj_t *prev*) [inline, static]

Get the next OS device in the system.

**Returns:**

the first OS device if `prev` is `NULL`.

#### 22.23.1.4 static hwloc_obj_t hwloc_get_next_pcidev (hwloc_topology_t *topology*, hwloc_obj_t *prev*) [inline, static]

Get the next PCI device in the system.

**Returns:**

the first PCI device if `prev` is `NULL`.

#### 22.23.1.5 static hwloc_obj_t hwloc_get_non_io_ancestor_obj (hwloc_topology_t *topology*, hwloc_obj_t *ioobj*) [inline, static]

Get the first non-I/O ancestor object. Given the I/O object `ioobj`, find the smallest non-I/O ancestor object. This object (normal or memory) may then be used for binding because it has non-NULL CPU and node sets and because its locality is the same as `ioobj`.

**Note:**

The resulting object is usually a normal object but it could also be a memory object (e.g. NUMA node) in future platforms if I/O objects ever get attached to memory instead of CPUs.

#### 22.23.1.6 static hwloc_obj_t hwloc_get_pcidev_by_busid (hwloc_topology_t *topology*, unsigned *domain*, unsigned *bus*, unsigned *dev*, unsigned *func*) [inline, static]

Find the PCI device object matching the PCI bus id given domain, bus device and function PCI bus id.

#### 22.23.1.7 static hwloc_obj_t hwloc_get_pcidev_by_busidstring (hwloc_topology_t *topology*, const char * *busid*) [inline, static]

Find the PCI device object matching the PCI bus id given as a string xxxx:yy:zz.t or yy:zz.t.

## 22.24 The bitmap API

### Defines

- #define hwloc_bitmap_foreach_begin(id, bitmap)
- #define hwloc_bitmap_foreach_end()

### Typedefs

- typedef struct hwloc_bitmap_s * hwloc_bitmap_t
- typedef struct hwloc_bitmap_s * hwloc_const_bitmap_t

### Functions

- hwloc_bitmap_t hwloc_bitmap_alloc (void)
- hwloc_bitmap_t hwloc_bitmap_alloc_full (void)
- void hwloc_bitmap_free (hwloc_bitmap_t bitmap)
- hwloc_bitmap_t hwloc_bitmap_dup (hwloc_const_bitmap_t bitmap)
- int hwloc_bitmap_copy (hwloc_bitmap_t dst, hwloc_const_bitmap_t src)
- int hwloc_bitmap_snprintf (char *restrict buf, size_t buflen, hwloc_const_bitmap_t bitmap)
- int hwloc_bitmap_asprintf (char **strp, hwloc_const_bitmap_t bitmap)
- int hwloc_bitmap_sscanf (hwloc_bitmap_t bitmap, const char *restrict string)
- int hwloc_bitmap_list_snprintf (char *restrict buf, size_t buflen, hwloc_const_bitmap_t bitmap)
- int hwloc_bitmap_list_asprintf (char **strp, hwloc_const_bitmap_t bitmap)
- int hwloc_bitmap_list_sscanf (hwloc_bitmap_t bitmap, const char *restrict string)
- int hwloc_bitmap_taskset_snprintf (char *restrict buf, size_t buflen, hwloc_const_bitmap_t bitmap)
- int hwloc_bitmap_taskset_asprintf (char **strp, hwloc_const_bitmap_t bitmap)
- int hwloc_bitmap_taskset_sscanf (hwloc_bitmap_t bitmap, const char *restrict string)
- void hwloc_bitmap_zero (hwloc_bitmap_t bitmap)
- void hwloc_bitmap_fill (hwloc_bitmap_t bitmap)
- int hwloc_bitmap_only (hwloc_bitmap_t bitmap, unsigned id)
- int hwloc_bitmap_allbut (hwloc_bitmap_t bitmap, unsigned id)
- int hwloc_bitmap_from_ulong (hwloc_bitmap_t bitmap, unsigned long mask)
- int hwloc_bitmap_from_ith_ulong (hwloc_bitmap_t bitmap, unsigned i, unsigned long mask)
- int hwloc_bitmap_from_ulongs (hwloc_bitmap_t bitmap, unsigned nr, const unsigned long *masks)
- int hwloc_bitmap_set (hwloc_bitmap_t bitmap, unsigned id)
- int hwloc_bitmap_set_range (hwloc_bitmap_t bitmap, unsigned begin, int end)
- int hwloc_bitmap_set_ith_ulong (hwloc_bitmap_t bitmap, unsigned i, unsigned long mask)
- int hwloc_bitmap_clr (hwloc_bitmap_t bitmap, unsigned id)
- int hwloc_bitmap_clr_range (hwloc_bitmap_t bitmap, unsigned begin, int end)
- int hwloc_bitmap_singlify (hwloc_bitmap_t bitmap)
- unsigned long hwloc_bitmap_to_ulong (hwloc_const_bitmap_t bitmap)
- unsigned long hwloc_bitmap_to_ith_ulong (hwloc_const_bitmap_t bitmap, unsigned i)
- int hwloc_bitmap_to_ulongs (hwloc_const_bitmap_t bitmap, unsigned nr, unsigned long *masks)
- int hwloc_bitmap_nr_ulongs (hwloc_const_bitmap_t bitmap)
- int hwloc_bitmap_isset (hwloc_const_bitmap_t bitmap, unsigned id)
- int hwloc_bitmap_iszero (hwloc_const_bitmap_t bitmap)
- int hwloc_bitmap_isfull (hwloc_const_bitmap_t bitmap)
- int hwloc_bitmap_first (hwloc_const_bitmap_t bitmap)

- int hwloc_bitmap_next (hwloc_const_bitmap_t bitmap, int prev)
- int hwloc_bitmap_last (hwloc_const_bitmap_t bitmap)
- int hwloc_bitmap_weight (hwloc_const_bitmap_t bitmap)
- int hwloc_bitmap_first_unset (hwloc_const_bitmap_t bitmap)
- int hwloc_bitmap_next_unset (hwloc_const_bitmap_t bitmap, int prev)
- int hwloc_bitmap_last_unset (hwloc_const_bitmap_t bitmap)
- int hwloc_bitmap_or (hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)
- int hwloc_bitmap_and (hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)
- int hwloc_bitmap_andnot (hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)
- int hwloc_bitmap_xor (hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)
- int hwloc_bitmap_not (hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap)
- int hwloc_bitmap_intersects (hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)
- int hwloc_bitmap_isincluded (hwloc_const_bitmap_t sub_bitmap, hwloc_const_bitmap_t super_bitmap)
- int hwloc_bitmap_isequal (hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)
- int hwloc_bitmap_compare_first (hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)
- int hwloc_bitmap_compare (hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)

### 22.24.1 Detailed Description

The hwloc_bitmap_t type represents a set of integers (positive or null). A bitmap may be of infinite size (all bits are set after some point). A bitmap may even be full if all bits are set.

Bitmaps are used by hwloc for sets of OS processors (which may actually be hardware threads) as by hwloc_cpuset_t (a typedef for hwloc_bitmap_t), or sets of NUMA memory nodes as hwloc_nodeset_t (also a typedef for hwloc_bitmap_t). Those are used for cpuset and nodeset fields in the hwloc_obj structure, see Object Sets (hwloc_cpuset_t and hwloc_nodeset_t).

*Both CPU and node sets are always indexed by OS physical number.* However users should usually not build CPU and node sets manually (e.g. with hwloc_bitmap_set()). One should rather use existing object sets and combine them with hwloc_bitmap_or(), etc. For instance, binding the current thread on a pair of cores may be performed with:

```
hwloc_obj_t core1 = ... , core2 = ... ;
hwloc_bitmap_t set = hwloc_bitmap_alloc();
hwloc_bitmap_or(set, core1->cpuset, core2->cpuset);
hwloc_set_cpubind(topology, set, HWLOC_CPUBIND_THREAD);
hwloc_bitmap_free(set);
```

**Note:**

Most functions below return an int that may be negative in case of error. The usual error case would be an internal failure to realloc/extend the storage of the bitmap (`errno` would be set to `ENOMEM`). Several examples of using the bitmap API are available under the doc/examples/ directory in the source tree. Regression tests such as tests/hwloc/hwloc_bitmap∗.c also make intensive use of this API.

## 22.24.2 Define Documentation

### 22.24.2.1 #define hwloc_bitmap_foreach_begin(id, bitmap)

Loop macro iterating on bitmap `bitmap`. The loop must start with hwloc_bitmap_foreach_begin() and end with hwloc_bitmap_foreach_end() followed by a terminating ';'.

`index` is the loop variable; it should be an unsigned int. The first iteration will set `index` to the lowest index in the bitmap. Successive iterations will iterate through, in order, all remaining indexes set in the bitmap. To be specific: each iteration will return a value for `index` such that hwloc_bitmap_isset(bitmap, index) is true.

The assert prevents the loop from being infinite if the bitmap is infinitely set.

### 22.24.2.2 #define hwloc_bitmap_foreach_end()

End of loop macro iterating on a bitmap. Needs a terminating ';'.

**See also:**

hwloc_bitmap_foreach_begin()

## 22.24.3 Typedef Documentation

### 22.24.3.1 typedef struct hwloc_bitmap_s* hwloc_bitmap_t

Set of bits represented as an opaque pointer to an internal bitmap.

### 22.24.3.2 typedef struct hwloc_bitmap_s* hwloc_const_bitmap_t

a non-modifiable hwloc_bitmap_t

## 22.24.4 Function Documentation

### 22.24.4.1 int hwloc_bitmap_allbut (hwloc_bitmap_t *bitmap*, unsigned *id*)

Fill the bitmap `and` clear the index `id`.

### 22.24.4.2 hwloc_bitmap_t hwloc_bitmap_alloc (void)

Allocate a new empty bitmap.

**Returns:**

A valid bitmap or `NULL`.

The bitmap should be freed by a corresponding call to hwloc_bitmap_free().

### 22.24.4.3 hwloc_bitmap_t hwloc_bitmap_alloc_full (void)

Allocate a new full bitmap.

#### 22.24.4.4 int hwloc_bitmap_and (hwloc_bitmap_t *res*, hwloc_const_bitmap_t *bitmap1*, hwloc_const_bitmap_t *bitmap2*)

And bitmaps `bitmap1` and `bitmap2` and store the result in bitmap `res`. `res` can be the same as `bitmap1` or `bitmap2`

#### 22.24.4.5 int hwloc_bitmap_andnot (hwloc_bitmap_t *res*, hwloc_const_bitmap_t *bitmap1*, hwloc_const_bitmap_t *bitmap2*)

And bitmap `bitmap1` and the negation of `bitmap2` and store the result in bitmap `res`. `res` can be the same as `bitmap1` or `bitmap2`

#### 22.24.4.6 int hwloc_bitmap_asprintf (char ** *strp*, hwloc_const_bitmap_t *bitmap*)

Stringify a bitmap into a newly allocated string.

**Returns:**

-1 on error.

#### 22.24.4.7 int hwloc_bitmap_clr (hwloc_bitmap_t *bitmap*, unsigned *id*)

Remove index `id` from bitmap `bitmap`.

#### 22.24.4.8 int hwloc_bitmap_clr_range (hwloc_bitmap_t *bitmap*, unsigned *begin*, int *end*)

Remove indexes from `begin` to `end` in bitmap `bitmap`. If `end` is `-1`, the range is infinite.

#### 22.24.4.9 int hwloc_bitmap_compare (hwloc_const_bitmap_t *bitmap1*, hwloc_const_bitmap_t *bitmap2*)

Compare bitmaps `bitmap1` and `bitmap2` in lexicographic order. Lexicographic comparison of bitmaps, starting for their highest indexes. Compare last indexes first, then second, etc. The empty bitmap is considered lower than anything.

**Returns:**

-1 if `bitmap1` is considered smaller than `bitmap2`.
1 if `bitmap1` is considered larger than `bitmap2`.
0 if bitmaps are equal (contrary to hwloc_bitmap_compare_first()).

For instance comparing binary bitmaps 0011 and 0110 returns -1 (hence 0011 is considered smaller than 0110). Comparing 00101 and 01010 returns -1 too.

**Note:**

This is different from the non-existing hwloc_bitmap_compare_last() which would only compare the highest index of each bitmap.

#### 22.24.4.10 int hwloc_bitmap_compare_first (hwloc_const_bitmap_t *bitmap1*, hwloc_const_bitmap_t *bitmap2*)

Compare bitmaps `bitmap1` and `bitmap2` using their lowest index. A bitmap is considered smaller if its least significant bit is smaller. The empty bitmap is considered higher than anything (because its least significant bit does not exist).

**Returns:**

-1 if `bitmap1` is considered smaller than `bitmap2`.
1 if `bitmap1` is considered larger than `bitmap2`.

For instance comparing binary bitmaps 0011 and 0110 returns -1 (hence 0011 is considered smaller than 0110) because least significant bit of 0011 (0001) is smaller than least significant bit of 0110 (0010). Comparing 01001 and 00110 would also return -1 for the same reason.

**Returns:**

0 if bitmaps are considered equal, even if they are not strictly equal. They just need to have the same least significant bit. For instance, comparing binary bitmaps 0010 and 0110 returns 0 because they have the same least significant bit.

#### 22.24.4.11 int hwloc_bitmap_copy (hwloc_bitmap_t *dst*, hwloc_const_bitmap_t *src*)

Copy the contents of bitmap `src` into the already allocated bitmap `dst`.

#### 22.24.4.12 hwloc_bitmap_t hwloc_bitmap_dup (hwloc_const_bitmap_t *bitmap*)

Duplicate bitmap `bitmap` by allocating a new bitmap and copying `bitmap` contents. If `bitmap` is `NULL`, `NULL` is returned.

#### 22.24.4.13 void hwloc_bitmap_fill (hwloc_bitmap_t *bitmap*)

Fill bitmap `bitmap` with all possible indexes (even if those objects don't exist or are otherwise unavailable).

#### 22.24.4.14 int hwloc_bitmap_first (hwloc_const_bitmap_t *bitmap*)

Compute the first index (least significant bit) in bitmap `bitmap`.

**Returns:**

-1 if no index is set in `bitmap`.

#### 22.24.4.15 int hwloc_bitmap_first_unset (hwloc_const_bitmap_t *bitmap*)

Compute the first unset index (least significant bit) in bitmap `bitmap`.

**Returns:**

-1 if no index is unset in `bitmap`.

#### 22.24.4.16 void hwloc_bitmap_free (hwloc_bitmap_t *bitmap*)

Free bitmap `bitmap`. If `bitmap` is `NULL`, no operation is performed.

#### 22.24.4.17 int hwloc_bitmap_from_ith_ulong (hwloc_bitmap_t *bitmap*, unsigned *i*, unsigned long *mask*)

Setup bitmap `bitmap` from unsigned long `mask` used as `i` -th subset.

#### 22.24.4.18 int hwloc_bitmap_from_ulong (hwloc_bitmap_t *bitmap*, unsigned long *mask*)

Setup bitmap `bitmap` from unsigned long `mask`.

#### 22.24.4.19 int hwloc_bitmap_from_ulongs (hwloc_bitmap_t *bitmap*, unsigned *nr*, const unsigned long * *masks*)

Setup bitmap `bitmap` from unsigned longs `masks` used as first `nr` subsets.

#### 22.24.4.20 int hwloc_bitmap_intersects (hwloc_const_bitmap_t *bitmap1*, hwloc_const_bitmap_t *bitmap2*)

Test whether bitmaps `bitmap1` and `bitmap2` intersects.

**Returns:**

1 if bitmaps intersect, 0 otherwise.

#### 22.24.4.21 int hwloc_bitmap_isequal (hwloc_const_bitmap_t *bitmap1*, hwloc_const_bitmap_t *bitmap2*)

Test whether bitmap `bitmap1` is equal to bitmap `bitmap2`.

**Returns:**

1 if bitmaps are equal, 0 otherwise.

#### 22.24.4.22 int hwloc_bitmap_isfull (hwloc_const_bitmap_t *bitmap*)

Test whether bitmap `bitmap` is completely full.

**Returns:**

1 if bitmap is full, 0 otherwise.

**Note:**

A full bitmap is always infinitely set.

#### 22.24.4.23 int hwloc_bitmap_isincluded (hwloc_const_bitmap_t *sub_bitmap*, hwloc_const_bitmap_t *super_bitmap*)

Test whether bitmap `sub_bitmap` is part of bitmap `super_bitmap`.

**Returns:**

1 if `sub_bitmap` is included in `super_bitmap`, 0 otherwise.

**Note:**

The empty bitmap is considered included in any other bitmap.

#### 22.24.4.24 int hwloc_bitmap_isset (hwloc_const_bitmap_t *bitmap*, unsigned *id*)

Test whether index `id` is part of bitmap `bitmap`.

**Returns:**

1 if the bit at index `id` is set in bitmap `bitmap`, 0 otherwise.

#### 22.24.4.25 int hwloc_bitmap_iszero (hwloc_const_bitmap_t *bitmap*)

Test whether bitmap `bitmap` is empty.

**Returns:**

1 if bitmap is empty, 0 otherwise.

#### 22.24.4.26 int hwloc_bitmap_last (hwloc_const_bitmap_t *bitmap*)

Compute the last index (most significant bit) in bitmap `bitmap`.

**Returns:**

-1 if no index is set in `bitmap`, or if `bitmap` is infinitely set.

#### 22.24.4.27 int hwloc_bitmap_last_unset (hwloc_const_bitmap_t *bitmap*)

Compute the last unset index (most significant bit) in bitmap `bitmap`.

**Returns:**

-1 if no index is unset in `bitmap`, or if `bitmap` is infinitely set.

#### 22.24.4.28 int hwloc_bitmap_list_asprintf (char ** *strp*, hwloc_const_bitmap_t *bitmap*)

Stringify a bitmap into a newly allocated list string.

**Returns:**

-1 on error.

#### 22.24.4.29 int hwloc_bitmap_list_snprintf (char *restrict *buf*, size_t *buflen*, hwloc_const_bitmap_t *bitmap*)

Stringify a bitmap in the list format. Lists are comma-separated indexes or ranges. Ranges are dash separated indexes. The last range may not have an ending indexes if the bitmap is infinitely set.

Up to `buflen` characters may be written in buffer `buf`.

If `buflen` is 0, `buf` may safely be `NULL`.

**Returns:**

the number of character that were actually written if not truncating, or that would have been written (not including the ending \0).

#### 22.24.4.30 int hwloc_bitmap_list_sscanf (hwloc_bitmap_t *bitmap*, const char *restrict *string*)

Parse a list string and stores it in bitmap `bitmap`.

#### 22.24.4.31 int hwloc_bitmap_next (hwloc_const_bitmap_t *bitmap*, int *prev*)

Compute the next index in bitmap `bitmap` which is after index `prev`. If `prev` is -1, the first index is returned.

**Returns:**

-1 if no index with higher index is set in `bitmap`.

#### 22.24.4.32 int hwloc_bitmap_next_unset (hwloc_const_bitmap_t *bitmap*, int *prev*)

Compute the next unset index in bitmap `bitmap` which is after index `prev`. If `prev` is -1, the first unset index is returned.

**Returns:**

-1 if no index with higher index is unset in `bitmap`.

#### 22.24.4.33 int hwloc_bitmap_not (hwloc_bitmap_t *res*, hwloc_const_bitmap_t *bitmap*)

Negate bitmap `bitmap` and store the result in bitmap `res`. `res` can be the same as `bitmap`

#### 22.24.4.34 int hwloc_bitmap_nr_ulongs (hwloc_const_bitmap_t *bitmap*)

Return the number of unsigned longs required for storing bitmap `bitmap` entirely. This is the number of contiguous unsigned longs from the very first bit of the bitmap (even if unset) up to the last set bit. This is useful for knowing the `nr` parameter to pass to hwloc_bitmap_to_ulongs() (or which calls to hwloc_bitmap_to_ith_ulong() are needed) to entirely convert a bitmap into multiple unsigned longs.

When called on the output of hwloc_topology_get_topology_cpuset(), the returned number is large enough for all cpusets of the topology.

**Returns:**

-1 if `bitmap` is infinite.

#### 22.24.4.35 int hwloc_bitmap_only (hwloc_bitmap_t *bitmap*, unsigned *id*)

Empty the bitmap `bitmap` and add bit `id`.

#### 22.24.4.36 int hwloc_bitmap_or (hwloc_bitmap_t *res*, hwloc_const_bitmap_t *bitmap1*, hwloc_const_bitmap_t *bitmap2*)

Or bitmaps `bitmap1` and `bitmap2` and store the result in bitmap `res`. `res` can be the same as `bitmap1` or `bitmap2`

#### 22.24.4.37 int hwloc_bitmap_set (hwloc_bitmap_t *bitmap*, unsigned *id*)

Add index `id` in bitmap `bitmap`.

#### 22.24.4.38 int hwloc_bitmap_set_ith_ulong (hwloc_bitmap_t *bitmap*, unsigned *i*, unsigned long *mask*)

Replace `i` -th subset of bitmap `bitmap` with unsigned long `mask`.

#### 22.24.4.39 int hwloc_bitmap_set_range (hwloc_bitmap_t *bitmap*, unsigned *begin*, int *end*)

Add indexes from `begin` to `end` in bitmap `bitmap`. If `end` is `-1`, the range is infinite.

#### 22.24.4.40 int hwloc_bitmap_singlify (hwloc_bitmap_t *bitmap*)

Keep a single index among those set in bitmap `bitmap`. May be useful before binding so that the process does not have a chance of migrating between multiple logical CPUs in the original mask. Instead of running the task on any PU inside the given CPU set, the operating system scheduler will be forced to run it on a single of these PUs. It avoids a migration overhead and cache-line ping-pongs between PUs.

**Note:**

This function is NOT meant to distribute multiple processes within a single CPU set. It always return the same single bit when called multiple times on the same input set. hwloc_distrib() may be used for generating CPU sets to distribute multiple tasks below a single multi-PU object.
This function cannot be applied to an object set directly. It should be applied to a copy (which may be obtained with hwloc_bitmap_dup()).

#### 22.24.4.41 int hwloc_bitmap_snprintf (char *restrict *buf*, size_t *buflen*, hwloc_const_bitmap_t *bitmap*)

Stringify a bitmap. Up to `buflen` characters may be written in buffer `buf`.

If `buflen` is 0, `buf` may safely be `NULL`.

**Returns:**

the number of character that were actually written if not truncating, or that would have been written (not including the ending \0).

#### 22.24.4.42 int hwloc_bitmap_sscanf (hwloc_bitmap_t *bitmap*, const char *restrict *string*)

Parse a bitmap string and stores it in bitmap `bitmap`.

#### 22.24.4.43 int hwloc_bitmap_taskset_asprintf (char ** *strp*, hwloc_const_bitmap_t *bitmap*)

Stringify a bitmap into a newly allocated taskset-specific string.

**Returns:**

-1 on error.

#### 22.24.4.44 int hwloc_bitmap_taskset_snprintf (char *restrict *buf*, size_t *buflen*, hwloc_const_bitmap_t *bitmap*)

Stringify a bitmap in the taskset-specific format. The taskset command manipulates bitmap strings that contain a single (possible very long) hexadecimal number starting with 0x.

Up to `buflen` characters may be written in buffer `buf`.

If `buflen` is 0, `buf` may safely be `NULL`.

**Returns:**

the number of character that were actually written if not truncating, or that would have been written (not including the ending \0).

#### 22.24.4.45 int hwloc_bitmap_taskset_sscanf (hwloc_bitmap_t *bitmap*, const char *restrict *string*)

Parse a taskset-specific bitmap string and stores it in bitmap `bitmap`.

#### 22.24.4.46 unsigned long hwloc_bitmap_to_ith_ulong (hwloc_const_bitmap_t *bitmap*, unsigned *i*)

Convert the `i` -th subset of bitmap `bitmap` into unsigned long mask.

#### 22.24.4.47 unsigned long hwloc_bitmap_to_ulong (hwloc_const_bitmap_t *bitmap*)

Convert the beginning part of bitmap `bitmap` into unsigned long `mask`.

#### 22.24.4.48 int hwloc_bitmap_to_ulongs (hwloc_const_bitmap_t *bitmap*, unsigned *nr*, unsigned long * *masks*)

Convert the first `nr` subsets of bitmap `bitmap` into the array of `nr` unsigned long `masks`. `nr` may be determined earlier with hwloc_bitmap_nr_ulongs().

**Returns:**

0

#### 22.24.4.49 int hwloc_bitmap_weight (hwloc_const_bitmap_t *bitmap*)

Compute the "weight" of bitmap `bitmap` (i.e., number of indexes that are in the bitmap).

**Returns:**

the number of indexes that are in the bitmap.
-1 if `bitmap` is infinitely set.

#### 22.24.4.50 int hwloc_bitmap_xor (hwloc_bitmap_t *res*, hwloc_const_bitmap_t *bitmap1*, hwloc_const_bitmap_t *bitmap2*)

Xor bitmaps `bitmap1` and `bitmap2` and store the result in bitmap `res`. `res` can be the same as `bitmap1` or `bitmap2`

#### 22.24.4.51 void hwloc_bitmap_zero (hwloc_bitmap_t *bitmap*)

Empty the bitmap `bitmap`.

# 22.25 Exporting Topologies to XML

## Enumerations

- enum hwloc_topology_export_xml_flags_e { HWLOC_TOPOLOGY_EXPORT_XML_FLAG_V1 }

## Functions

- int hwloc_topology_export_xml (hwloc_topology_t topology, const char *xmlpath, unsigned long flags)
- int hwloc_topology_export_xmlbuffer (hwloc_topology_t topology, char **xmlbuffer, int *buflen, unsigned long flags)
- void hwloc_free_xmlbuffer (hwloc_topology_t topology, char *xmlbuffer)
- void hwloc_topology_set_userdata_export_callback (hwloc_topology_t topology, void(*export_cb)(void *reserved, hwloc_topology_t topology, hwloc_obj_t obj))
- int hwloc_export_obj_userdata (void *reserved, hwloc_topology_t topology, hwloc_obj_t obj, const char *name, const void *buffer, size_t length)
- int hwloc_export_obj_userdata_base64 (void *reserved, hwloc_topology_t topology, hwloc_obj_t obj, const char *name, const void *buffer, size_t length)
- void hwloc_topology_set_userdata_import_callback (hwloc_topology_t topology, void(*import_cb)(hwloc_topology_t topology, hwloc_obj_t obj, const char *name, const void *buffer, size_t length))

## 22.25.1 Enumeration Type Documentation

### 22.25.1.1 enum hwloc_topology_export_xml_flags_e

Flags for exporting XML topologies. Flags to be given as a OR'ed set to hwloc_topology_export_xml().

**Enumerator:**

***HWLOC_TOPOLOGY_EXPORT_XML_FLAG_V1*** Export XML that is loadable by hwloc v1.x. However, the export may miss some details about the topology.

## 22.25.2 Function Documentation

### 22.25.2.1 int hwloc_export_obj_userdata (void * *reserved*, hwloc_topology_t *topology*, hwloc_obj_t *obj*, const char * *name*, const void * *buffer*, size_t *length*)

Export some object userdata to XML. This function may only be called from within the export() callback passed to hwloc_topology_set_userdata_export_callback(). It may be invoked one of multiple times to export some userdata to XML. The `buffer` content of length `length` is stored with optional name `name`.

When importing this XML file, the import() callback (if set) will be called exactly as many times as hwloc_export_obj_userdata() was called during export(). It will receive the corresponding `name`, `buffer` and `length` arguments.

`reserved`, `topology` and `obj` must be the first three parameters that were given to the export callback.

Only printable characters may be exported to XML string attributes. If a non-printable character is passed in `name` or `buffer`, the function returns -1 with errno set to EINVAL.

If exporting binary data, the application should first encode into printable characters only (or use hwloc_-export_obj_userdata_base64()). It should also take care of portability issues if the export may be reimported on a different architecture.

#### 22.25.2.2 int hwloc_export_obj_userdata_base64 (void * *reserved*, hwloc_topology_t *topology*, hwloc_obj_t *obj*, const char * *name*, const void * *buffer*, size_t *length*)

Encode and export some object userdata to XML. This function is similar to hwloc_export_obj_userdata() but it encodes the input buffer into printable characters before exporting. On import, decoding is automatically performed before the data is given to the import() callback if any.

This function may only be called from within the export() callback passed to hwloc_topology_set_-userdata_export_callback().

The function does not take care of portability issues if the export may be reimported on a different architecture.

#### 22.25.2.3 void hwloc_free_xmlbuffer (hwloc_topology_t *topology*, char * *xmlbuffer*)

Free a buffer allocated by hwloc_topology_export_xmlbuffer().

#### 22.25.2.4 int hwloc_topology_export_xml (hwloc_topology_t *topology*, const char * *xmlpath*, unsigned long *flags*)

Export the topology into an XML file. This file may be loaded later through hwloc_topology_set_xml().

By default, the latest export format is used, which means older hwloc releases (e.g. v1.x) will not be able to import it. Exporting to v1.x specific XML format is possible using flag HWLOC_TOPOLOGY_-EXPORT_XML_FLAG_V1 but it may miss some details about the topology. If there is any chance that the exported file may ever be imported back by a process using hwloc 1.x, one should consider detecting it at runtime and using the corresponding export format.

`flags` is a OR'ed set of hwloc_topology_export_xml_flags_e.

**Returns:**

-1 if a failure occured.

**Note:**

See also hwloc_topology_set_userdata_export_callback() for exporting application-specific object userdata.
The topology-specific userdata pointer is ignored when exporting to XML.
Only printable characters may be exported to XML string attributes. Any other character, especially any non-ASCII character, will be silently dropped.
If `name` is "-", the XML output is sent to the standard output.

#### 22.25.2.5 int hwloc_topology_export_xmlbuffer (hwloc_topology_t *topology*, char ** *xmlbuffer*, int * *buflen*, unsigned long *flags*)

Export the topology into a newly-allocated XML memory buffer. `xmlbuffer` is allocated by the callee and should be freed with hwloc_free_xmlbuffer() later in the caller.

This memory buffer may be loaded later through hwloc_topology_set_xmlbuffer().

By default, the latest export format is used, which means older hwloc releases (e.g. v1.x) will not be able to import it. Exporting to v1.x specific XML format is possible using flag HWLOC_TOPOLOGY_-EXPORT_XML_FLAG_V1 but it may miss some details about the topology. If there is any chance that the exported buffer may ever be imported back by a process using hwloc 1.x, one should consider detecting it at runtime and using the corresponding export format.

The returned buffer ends with a that is included in the returned length.

`flags` is a OR'ed set of hwloc_topology_export_xml_flags_e.

**Returns:**

-1 if a failure occured.

**Note:**

See also hwloc_topology_set_userdata_export_callback() for exporting application-specific object userdata.
The topology-specific userdata pointer is ignored when exporting to XML.
Only printable characters may be exported to XML string attributes. Any other character, especially any non-ASCII character, will be silently dropped.

#### 22.25.2.6 void hwloc_topology_set_userdata_export_callback (hwloc_topology_t *topology*, void(∗)(void ∗reserved, hwloc_topology_t topology, hwloc_obj_t obj) *export_cb*)

Set the application-specific callback for exporting object userdata. The object userdata pointer is not exported to XML by default because hwloc does not know what it contains.

This function lets applications set `export_cb` to a callback function that converts this opaque userdata into an exportable string.

`export_cb` is invoked during XML export for each object whose `userdata` pointer is not `NULL`. The callback should use hwloc_export_obj_userdata() or hwloc_export_obj_userdata_base64() to actually export something to XML (possibly multiple times per object).

`export_cb` may be set to `NULL` if userdata should not be exported to XML.

**Note:**

The topology-specific userdata pointer is ignored when exporting to XML.

#### 22.25.2.7 void hwloc_topology_set_userdata_import_callback (hwloc_topology_t *topology*, void(∗)(hwloc_topology_t topology, hwloc_obj_t obj, const char ∗name, const void ∗buffer, size_t length) *import_cb*)

Set the application-specific callback for importing userdata. On XML import, userdata is ignored by default because hwloc does not know how to store it in memory.

This function lets applications set `import_cb` to a callback function that will get the XML-stored userdata and store it in the object as expected by the application.

`import_cb` is called during hwloc_topology_load() as many times as hwloc_export_obj_userdata() was called during export. The topology is not entirely setup yet. Object attributes are ready to consult, but links between objects are not.

`import_cb` may be `NULL` if userdata should be ignored during import.

**Note:**

`buffer` contains `length` characters followed by a null byte (”).
This function should be called before hwloc_topology_load().
The topology-specific userdata pointer is ignored when importing from XML.

# 22.26 Exporting Topologies to Synthetic

## Enumerations

- enum hwloc_topology_export_synthetic_flags_e { HWLOC_TOPOLOGY_EXPORT_SYNTHETIC_FLAG_NO_EXTENDED_TYPES, HWLOC_TOPOLOGY_EXPORT_SYNTHETIC_FLAG_NO_ATTRS, HWLOC_TOPOLOGY_EXPORT_SYNTHETIC_FLAG_V1, HWLOC_TOPOLOGY_EXPORT_SYNTHETIC_FLAG_IGNORE_MEMORY }

## Functions

- int hwloc_topology_export_synthetic (hwloc_topology_t topology, char *buffer, size_t buflen, unsigned long flags)

## 22.26.1 Enumeration Type Documentation

### 22.26.1.1 enum hwloc_topology_export_synthetic_flags_e

Flags for exporting synthetic topologies. Flags to be given as a OR'ed set to hwloc_topology_export_synthetic().

**Enumerator:**

***HWLOC_TOPOLOGY_EXPORT_SYNTHETIC_FLAG_NO_EXTENDED_TYPES*** Export extended types such as L2dcache as basic types such as Cache. This is required if loading the synthetic description with hwloc $< 1.9$.

***HWLOC_TOPOLOGY_EXPORT_SYNTHETIC_FLAG_NO_ATTRS*** Do not export level attributes. Ignore level attributes such as memory/cache sizes or PU indexes. This is required if loading the synthetic description with hwloc $< 1.10$.

***HWLOC_TOPOLOGY_EXPORT_SYNTHETIC_FLAG_V1*** Export the memory hierarchy as expected in hwloc 1.x. Instead of attaching memory children to levels, export single NUMA node child as normal intermediate levels, when possible. This is required if loading the synthetic description with hwloc 1.x. However this may fail if some objects have multiple local NUMA nodes.

***HWLOC_TOPOLOGY_EXPORT_SYNTHETIC_FLAG_IGNORE_MEMORY*** Do not export memory information. Only export the actual hierarchy of normal CPU-side objects and ignore where memory is attached. This is useful for when the hierarchy of CPUs is what really matters, but it behaves as if there was a single machine-wide NUMA node.

## 22.26.2 Function Documentation

### 22.26.2.1 int hwloc_topology_export_synthetic (hwloc_topology_t *topology*, char * *buffer*, size_t *buflen*, unsigned long *flags*)

Export the topology as a synthetic string. At most `buflen` characters will be written in `buffer`, including the terminating .

This exported string may be given back to hwloc_topology_set_synthetic().

`flags` is a OR'ed set of hwloc_topology_export_synthetic_flags_e.

**Returns:**

The number of characters that were written, not including the terminating .
-1 if the topology could not be exported, for instance if it is not symmetric.

**Note:**

I/O and Misc children are ignored, the synthetic string only describes normal children.
A 1024-byte buffer should be large enough for exporting topologies in the vast majority of cases.

# 22.27 Retrieve distances between objects

## Data Structures

- struct hwloc_distances_s

  *Matrix of distances between a set of objects.*

## Enumerations

- enum hwloc_distances_kind_e {

  HWLOC_DISTANCES_KIND_FROM_OS, HWLOC_DISTANCES_KIND_FROM_USER, HWLOC_DISTANCES_KIND_MEANS_LATENCY, HWLOC_DISTANCES_KIND_MEANS_-BANDWIDTH,

  HWLOC_DISTANCES_KIND_HETEROGENEOUS_TYPES }

## Functions

- int hwloc_distances_get (hwloc_topology_t topology, unsigned *nr, struct hwloc_distances_s **distances, unsigned long kind, unsigned long flags)
- int hwloc_distances_get_by_depth (hwloc_topology_t topology, int depth, unsigned *nr, struct hwloc_distances_s **distances, unsigned long kind, unsigned long flags)
- int hwloc_distances_get_by_type (hwloc_topology_t topology, hwloc_obj_type_t type, unsigned *nr, struct hwloc_distances_s **distances, unsigned long kind, unsigned long flags)
- int hwloc_distances_get_by_name (hwloc_topology_t topology, const char *name, unsigned *nr, struct hwloc_distances_s **distances, unsigned long flags)
- const char * hwloc_distances_get_name (hwloc_topology_t topology, struct hwloc_distances_s *distances)
- void hwloc_distances_release (hwloc_topology_t topology, struct hwloc_distances_s *distances)

## 22.27.1 Enumeration Type Documentation

### 22.27.1.1 enum hwloc_distances_kind_e

Kinds of distance matrices. The `kind` attribute of struct hwloc_distances_s is a OR'ed set of kinds.

A kind of format HWLOC_DISTANCES_KIND_FROM_* specifies where the distance information comes from, if known.

A kind of format HWLOC_DISTANCES_KIND_MEANS_* specifies whether values are latencies or bandwidths, if applicable.

**Enumerator:**

***HWLOC_DISTANCES_KIND_FROM_OS*** These distances were obtained from the operating system or hardware.

***HWLOC_DISTANCES_KIND_FROM_USER*** These distances were provided by the user.

***HWLOC_DISTANCES_KIND_MEANS_LATENCY*** Distance values are similar to latencies between objects. Values are smaller for closer objects, hence minimal on the diagonal of the matrix (distance between an object and itself). It could also be the number of network hops between objects, etc.

***HWLOC_DISTANCES_KIND_MEANS_BANDWIDTH*** Distance values are similar to bandwidths between objects. Values are higher for closer objects, hence maximal on the diagonal of the matrix (distance between an object and itself). Such values are currently ignored for distance-based grouping.

***HWLOC_DISTANCES_KIND_HETEROGENEOUS_TYPES*** This distances structure covers objects of different types.

## 22.27.2 Function Documentation

### 22.27.2.1 int hwloc_distances_get (hwloc_topology_t *topology*, unsigned * *nr*, struct hwloc_distances_s ** *distances*, unsigned long *kind*, unsigned long *flags*)

Retrieve distance matrices. Retrieve distance matrices from the topology into the `distances` array.

`flags` is currently unused, should be `0`.

`kind` serves as a filter. If `0`, all distance matrices are returned. If it contains some HWLOC_-DISTANCES_KIND_FROM_*, only distance matrices whose kind matches one of these are returned. If it contains some HWLOC_DISTANCES_KIND_MEANS_*, only distance matrices whose kind matches one of these are returned.

On input, `nr` points to the number of distance matrices that may be stored in `distances`. On output, `nr` points to the number of distance matrices that were actually found, even if some of them couldn't be stored in `distances`. Distance matrices that couldn't be stored are ignored, but the function still returns success (`0`). The caller may find out by comparing the value pointed by `nr` before and after the function call.

Each distance matrix returned in the `distances` array should be released by the caller using hwloc_-distances_release().

### 22.27.2.2 int hwloc_distances_get_by_depth (hwloc_topology_t *topology*, int *depth*, unsigned * *nr*, struct hwloc_distances_s ** *distances*, unsigned long *kind*, unsigned long *flags*)

Retrieve distance matrices for object at a specific depth in the topology. Identical to hwloc_distances_get() with the additional `depth` filter.

### 22.27.2.3 int hwloc_distances_get_by_name (hwloc_topology_t *topology*, const char * *name*, unsigned * *nr*, struct hwloc_distances_s ** *distances*, unsigned long *flags*)

Retrieve a distance matrix with the given name. Usually only one distances structure may match a given name.

### 22.27.2.4 int hwloc_distances_get_by_type (hwloc_topology_t *topology*, hwloc_obj_type_t *type*, unsigned * *nr*, struct hwloc_distances_s ** *distances*, unsigned long *kind*, unsigned long *flags*)

Retrieve distance matrices for object of a specific type. Identical to hwloc_distances_get() with the additional `type` filter.

#### 22.27.2.5 const char* hwloc_distances_get_name (hwloc_topology_t *topology*, struct hwloc_distances_s * *distances*)

Get a description of what a distances structure contains. For instance "NUMALatency" for hardware-provided NUMA distances (ACPI SLIT), or NULL if unknown.

#### 22.27.2.6 void hwloc_distances_release (hwloc_topology_t *topology*, struct hwloc_distances_s * *distances*)

Release a distance matrix structure previously returned by hwloc_distances_get().

**Note:**

This function is not required if the structure is removed with hwloc_distances_release_remove().

## 22.28 Helpers for consulting distance matrices

### Functions

- static int hwloc_distances_obj_index (struct hwloc_distances_s *distances, hwloc_obj_t obj)
- static int hwloc_distances_obj_pair_values (struct hwloc_distances_s *distances, hwloc_obj_t obj1, hwloc_obj_t obj2, hwloc_uint64_t *value1to2, hwloc_uint64_t *value2to1)

### 22.28.1 Function Documentation

#### 22.28.1.1 static int hwloc_distances_obj_index (struct hwloc_distances_s * *distances*, hwloc_obj_t *obj*) [inline, static]

Find the index of an object in a distances structure.

**Returns:**

-1 if object `obj` is not involved in structure `distances`.

#### 22.28.1.2 static int hwloc_distances_obj_pair_values (struct hwloc_distances_s * *distances*, hwloc_obj_t *obj1*, hwloc_obj_t *obj2*, hwloc_uint64_t * *value1to2*, hwloc_uint64_t * *value2to1*) [inline, static]

Find the values between two objects in a distance matrices. The distance from `obj1` to `obj2` is stored in the value pointed by `value1to2` and reciprocally.

**Returns:**

-1 if object `obj1` or `obj2` is not involved in structure `distances`.

# 22.29 Add or remove distances between objects

## Enumerations

- enum hwloc_distances_add_flag_e { HWLOC_DISTANCES_ADD_FLAG_GROUP, HWLOC_-DISTANCES_ADD_FLAG_GROUP_INACCURATE }

## Functions

- int hwloc_distances_add (hwloc_topology_t topology, unsigned nbobjs, hwloc_obj_t *objs, hwloc_-uint64_t *values, unsigned long kind, unsigned long flags)
- int hwloc_distances_remove (hwloc_topology_t topology)
- int hwloc_distances_remove_by_depth (hwloc_topology_t topology, int depth)
- static int hwloc_distances_remove_by_type (hwloc_topology_t topology, hwloc_obj_type_t type)
- int hwloc_distances_release_remove (hwloc_topology_t topology, struct hwloc_distances_s *distances)

## 22.29.1 Enumeration Type Documentation

### 22.29.1.1 enum hwloc_distances_add_flag_e

Flags for adding a new distances to a topology.

**Enumerator:**

***HWLOC_DISTANCES_ADD_FLAG_GROUP*** Try to group objects based on the newly provided distance information.

***HWLOC_DISTANCES_ADD_FLAG_GROUP_INACCURATE*** If grouping, consider the distance values as inaccurate and relax the comparisons during the grouping algorithms. The actual accuracy may be modified through the HWLOC_GROUPING_ACCURACY environment variable (see Environment Variables).

## 22.29.2 Function Documentation

### 22.29.2.1 int hwloc_distances_add (hwloc_topology_t *topology*, unsigned *nbobjs*, hwloc_obj_t * *objs*, hwloc_uint64_t * *values*, unsigned long *kind*, unsigned long *flags*)

Provide a new distance matrix. Provide the matrix of distances between a set of objects given by `nbobjs` and the `objs` array. `nbobjs` must be at least 2. The distances are stored as a one-dimension array in `values`. The distance from object i to object j is in slot i*nbobjs+j.

`kind` specifies the kind of distance as a OR'ed set of hwloc_distances_kind_e. Kind HWLOC_-DISTANCES_KIND_HETEROGENEOUS_TYPES will be automatically added if objects of different types are given.

`flags` configures the behavior of the function using an optional OR'ed set of hwloc_distances_add_flag_-e.

#### 22.29.2.2 int hwloc_distances_release_remove (hwloc_topology_t *topology*, struct hwloc_distances_s * *distances*)

Release and remove the given distance matrice from the topology. This function includes a call to hwloc_distances_release().

#### 22.29.2.3 int hwloc_distances_remove (hwloc_topology_t *topology*)

Remove all distance matrices from a topology. Remove all distance matrices, either provided by the user or gathered through the OS.

If these distances were used to group objects, these additional Group objects are not removed from the topology.

#### 22.29.2.4 int hwloc_distances_remove_by_depth (hwloc_topology_t *topology*, int *depth*)

Remove distance matrices for objects at a specific depth in the topology. Identical to hwloc_distances_remove() but only applies to one level of the topology.

#### 22.29.2.5 static int hwloc_distances_remove_by_type (hwloc_topology_t *topology*, hwloc_obj_type_t *type*) [inline, static]

Remove distance matrices for objects of a specific type in the topology. Identical to hwloc_distances_remove() but only applies to one level of the topology.

# 22.30 Linux-specific helpers

## Functions

- int hwloc_linux_set_tid_cpubind (hwloc_topology_t topology, pid_t tid, hwloc_const_cpuset_t set)
- int hwloc_linux_get_tid_cpubind (hwloc_topology_t topology, pid_t tid, hwloc_cpuset_t set)
- int hwloc_linux_get_tid_last_cpu_location (hwloc_topology_t topology, pid_t tid, hwloc_bitmap_t set)
- int hwloc_linux_read_path_as_cpumask (const char *path, hwloc_bitmap_t set)

## 22.30.1 Detailed Description

This includes helpers for manipulating Linux kernel cpumap files, and hwloc equivalents of the Linux sched_setaffinity and sched_getaffinity system calls.

## 22.30.2 Function Documentation

### 22.30.2.1 int hwloc_linux_get_tid_cpubind (hwloc_topology_t *topology*, pid_t *tid*, hwloc_cpuset_t *set*)

Get the current binding of thread `tid`. The behavior is exactly the same as the Linux sched_getaffinity system call, but uses a hwloc cpuset.

**Note:**

This is equivalent to calling hwloc_get_proc_cpubind() with HWLOC_CPUBIND_THREAD as flags.

### 22.30.2.2 int hwloc_linux_get_tid_last_cpu_location (hwloc_topology_t *topology*, pid_t *tid*, hwloc_bitmap_t *set*)

Get the last physical CPU where thread `tid` ran.

**Note:**

This is equivalent to calling hwloc_get_proc_last_cpu_location() with HWLOC_CPUBIND_THREAD as flags.

### 22.30.2.3 int hwloc_linux_read_path_as_cpumask (const char * *path*, hwloc_bitmap_t *set*)

Convert a linux kernel cpumask file `path` into a hwloc bitmap `set`. Might be used when reading CPU set from sysfs attributes such as topology and caches for processors, or local_cpus for devices.

**Note:**

This function ignores the HWLOC_FSROOT environment variable.

#### 22.30.2.4 int hwloc_linux_set_tid_cpubind (hwloc_topology_t *topology*, pid_t *tid*, hwloc_const_cpuset_t *set*)

Bind a thread `tid` on cpus given in cpuset `set`. The behavior is exactly the same as the Linux sched_-setaffinity system call, but uses a hwloc cpuset.

**Note:**

This is equivalent to calling hwloc_set_proc_cpubind() with HWLOC_CPUBIND_THREAD as flags.

# 22.31 Interoperability with Linux libnuma unsigned long masks

## Functions

- static int hwloc_cpuset_to_linux_libnuma_ulongs (hwloc_topology_t topology, hwloc_const_cpuset_t cpuset, unsigned long ∗mask, unsigned long ∗maxnode)
- static int hwloc_nodeset_to_linux_libnuma_ulongs (hwloc_topology_t topology, hwloc_const_nodeset_t nodeset, unsigned long ∗mask, unsigned long ∗maxnode)
- static int hwloc_cpuset_from_linux_libnuma_ulongs (hwloc_topology_t topology, hwloc_cpuset_t cpuset, const unsigned long ∗mask, unsigned long maxnode)
- static int hwloc_nodeset_from_linux_libnuma_ulongs (hwloc_topology_t topology, hwloc_nodeset_t nodeset, const unsigned long ∗mask, unsigned long maxnode)

### 22.31.1 Detailed Description

This interface helps converting between Linux libnuma unsigned long masks and hwloc cpusets and nodesets.

**Note:**

Topology `topology` must match the current machine.
The behavior of libnuma is undefined if the kernel is not NUMA-aware. (when CONFIG_NUMA is not set in the kernel configuration). This helper and libnuma may thus not be strictly compatible in this case, which may be detected by checking whether numa_available() returns -1.

### 22.31.2 Function Documentation

#### 22.31.2.1 static int hwloc_cpuset_from_linux_libnuma_ulongs (hwloc_topology_t *topology*, hwloc_cpuset_t *cpuset*, const unsigned long ∗ *mask*, unsigned long *maxnode*) [inline, static]

Convert the array of unsigned long `mask` into hwloc CPU set. `mask` is a array of unsigned long that will be read. `maxnode` contains the maximal node number that may be read in `mask`.

This function may be used after calling get_mempolicy or any other function that takes an array of unsigned long as output parameter (and possibly a maximal node number as input parameter).

#### 22.31.2.2 static int hwloc_cpuset_to_linux_libnuma_ulongs (hwloc_topology_t *topology*, hwloc_const_cpuset_t *cpuset*, unsigned long ∗ *mask*, unsigned long ∗ *maxnode*) [inline, static]

Convert hwloc CPU set `cpuset` into the array of unsigned long `mask`. `mask` is the array of unsigned long that will be filled. `maxnode` contains the maximal node number that may be stored in `mask`. `maxnode` will be set to the maximal node number that was found, plus one.

This function may be used before calling set_mempolicy, mbind, migrate_pages or any other function that takes an array of unsigned long and a maximal node number as input parameter.

#### 22.31.2.3 static int hwloc_nodeset_from_linux_libnuma_ulongs (hwloc_topology_t *topology*, hwloc_nodeset_t *nodeset*, const unsigned long * *mask*, unsigned long *maxnode*) [inline, static]

Convert the array of unsigned long `mask` into hwloc NUMA node set. `mask` is a array of unsigned long that will be read. `maxnode` contains the maximal node number that may be read in `mask`.

This function may be used after calling get_mempolicy or any other function that takes an array of unsigned long as output parameter (and possibly a maximal node number as input parameter).

#### 22.31.2.4 static int hwloc_nodeset_to_linux_libnuma_ulongs (hwloc_topology_t *topology*, hwloc_const_nodeset_t *nodeset*, unsigned long * *mask*, unsigned long * *maxnode*) [inline, static]

Convert hwloc NUMA node set `nodeset` into the array of unsigned long `mask`. `mask` is the array of unsigned long that will be filled. `maxnode` contains the maximal node number that may be stored in `mask`. `maxnode` will be set to the maximal node number that was found, plus one.

This function may be used before calling set_mempolicy, mbind, migrate_pages or any other function that takes an array of unsigned long and a maximal node number as input parameter.

# 22.32 Interoperability with Linux libnuma bitmask

## Functions

- static struct bitmask * hwloc_cpuset_to_linux_libnuma_bitmask (hwloc_topology_t topology, hwloc_const_cpuset_t cpuset)
- static struct bitmask * hwloc_nodeset_to_linux_libnuma_bitmask (hwloc_topology_t topology, hwloc_const_nodeset_t nodeset)
- static int hwloc_cpuset_from_linux_libnuma_bitmask (hwloc_topology_t topology, hwloc_cpuset_t cpuset, const struct bitmask *bitmask)
- static int hwloc_nodeset_from_linux_libnuma_bitmask (hwloc_topology_t topology, hwloc_nodeset_t nodeset, const struct bitmask *bitmask)

## 22.32.1 Detailed Description

This interface helps converting between Linux libnuma bitmasks and hwloc cpusets and nodesets.

**Note:**

Topology `topology` must match the current machine.
The behavior of libnuma is undefined if the kernel is not NUMA-aware. (when CONFIG_NUMA is not set in the kernel configuration). This helper and libnuma may thus not be strictly compatible in this case, which may be detected by checking whether numa_available() returns -1.

## 22.32.2 Function Documentation

### 22.32.2.1 static int hwloc_cpuset_from_linux_libnuma_bitmask (hwloc_topology_t *topology*, hwloc_cpuset_t *cpuset*, const struct bitmask * *bitmask*) [inline, static]

Convert libnuma bitmask `bitmask` into hwloc CPU set `cpuset`. This function may be used after calling many numa_ functions that use a struct bitmask as an output parameter.

### 22.32.2.2 static struct bitmask * hwloc_cpuset_to_linux_libnuma_bitmask (hwloc_topology_t *topology*, hwloc_const_cpuset_t *cpuset*) [static, read]

Convert hwloc CPU set `cpuset` into the returned libnuma bitmask. The returned bitmask should later be freed with numa_bitmask_free.

This function may be used before calling many numa_ functions that use a struct bitmask as an input parameter.

**Returns:**

newly allocated struct bitmask.

### 22.32.2.3 static int hwloc_nodeset_from_linux_libnuma_bitmask (hwloc_topology_t *topology*, hwloc_nodeset_t *nodeset*, const struct bitmask * *bitmask*) [inline, static]

Convert libnuma bitmask `bitmask` into hwloc NUMA node set `nodeset`. This function may be used after calling many numa_ functions that use a struct bitmask as an output parameter.

#### 22.32.2.4 static struct bitmask * hwloc_nodeset_to_linux_libnuma_bitmask (hwloc_topology_t *topology*, hwloc_const_nodeset_t *nodeset*) [static, read]

Convert hwloc NUMA node set `nodeset` into the returned libnuma bitmask. The returned bitmask should later be freed with numa_bitmask_free.

This function may be used before calling many numa_ functions that use a struct bitmask as an input parameter.

**Returns:**

newly allocated struct bitmask.

# 22.33 Interoperability with glibc sched affinity

## Functions

- static int hwloc_cpuset_to_glibc_sched_affinity (hwloc_topology_t topology, hwloc_const_cpuset_t hwlocset, cpu_set_t ∗schedset, size_t schedsetsize)
- static int hwloc_cpuset_from_glibc_sched_affinity (hwloc_topology_t topology, hwloc_cpuset_t hwlocset, const cpu_set_t ∗schedset, size_t schedsetsize)

### 22.33.1 Detailed Description

This interface offers ways to convert between hwloc cpusets and glibc cpusets such as those manipulated by sched_getaffinity() or pthread_attr_setaffinity_np().

**Note:**

Topology `topology` must match the current machine.

### 22.33.2 Function Documentation

#### 22.33.2.1 static int hwloc_cpuset_from_glibc_sched_affinity (hwloc_topology_t *topology*, hwloc_cpuset_t *hwlocset*, const cpu_set_t ∗ *schedset*, size_t *schedsetsize*) [inline, static]

Convert glibc sched affinity CPU set `schedset` into hwloc CPU set. This function may be used before calling sched_setaffinity or any other function that takes a cpu_set_t as input parameter.

`schedsetsize` should be sizeof(cpu_set_t) unless `schedset` was dynamically allocated with CPU_-ALLOC

#### 22.33.2.2 static int hwloc_cpuset_to_glibc_sched_affinity (hwloc_topology_t *topology*, hwloc_const_cpuset_t *hwlocset*, cpu_set_t ∗ *schedset*, size_t *schedsetsize*) [inline, static]

Convert hwloc CPU set `toposet` into glibc sched affinity CPU set `schedset`. This function may be used before calling sched_setaffinity or any other function that takes a cpu_set_t as input parameter.

`schedsetsize` should be sizeof(cpu_set_t) unless `schedset` was dynamically allocated with CPU_-ALLOC

## 22.34 Interoperability with OpenCL

### Functions

- static int hwloc_opencl_get_device_pci_busid (cl_device_id device, unsigned *domain, unsigned *bus, unsigned *dev, unsigned *func)
- static int hwloc_opencl_get_device_cpuset (hwloc_topology_t topology, cl_device_id device, hwloc_cpuset_t set)
- static hwloc_obj_t hwloc_opencl_get_device_osdev_by_index (hwloc_topology_t topology, unsigned platform_index, unsigned device_index)
- static hwloc_obj_t hwloc_opencl_get_device_osdev (hwloc_topology_t topology, cl_device_id device)

### 22.34.1 Detailed Description

This interface offers ways to retrieve topology information about OpenCL devices.

Only AMD and NVIDIA OpenCL implementations currently offer useful locality information about their devices.

### 22.34.2 Function Documentation

#### 22.34.2.1 static int hwloc_opencl_get_device_cpuset (hwloc_topology_t *topology*, cl_device_id *device*, hwloc_cpuset_t *set*) [inline, static]

Get the CPU set of logical processors that are physically close to OpenCL device `device`. Return the CPU set describing the locality of the OpenCL device `device`.

Topology `topology` and device `device` must match the local machine. I/O devices detection and the OpenCL component are not needed in the topology.

The function only returns the locality of the device. If more information about the device is needed, OS objects should be used instead, see hwloc_opencl_get_device_osdev() and hwloc_opencl_get_device_osdev_by_index().

This function is currently only implemented in a meaningful way for Linux with the AMD or NVIDIA OpenCL implementation; other systems will simply get a full cpuset.

#### 22.34.2.2 static hwloc_obj_t hwloc_opencl_get_device_osdev (hwloc_topology_t *topology*, cl_device_id *device*) [inline, static]

Get the hwloc OS device object corresponding to OpenCL device `deviceX`. Use OpenCL device attributes to find the corresponding hwloc OS device object. Return NULL if there is none or if useful attributes are not available.

This function currently only works on AMD and NVIDIA OpenCL devices that support relevant OpenCL extensions. hwloc_opencl_get_device_osdev_by_index() should be preferred whenever possible, i.e. when platform and device index are known.

Topology `topology` and device `device` must match the local machine. I/O devices detection and the OpenCL component must be enabled in the topology. If not, the locality of the object may still be found using hwloc_opencl_get_device_cpuset().

**Note:**

This function cannot work if PCI devices are filtered out.
The corresponding hwloc PCI device may be found by looking at the result parent pointer (unless PCI devices are filtered out).

#### 22.34.2.3 static hwloc_obj_t hwloc_opencl_get_device_osdev_by_index (hwloc_topology_t *topology*, unsigned *platform_index*, unsigned *device_index*) [inline, static]

Get the hwloc OS device object corresponding to the OpenCL device for the given indexes. Return the OS device object describing the OpenCL device whose platform index is `platform_index`, and whose device index within this platform if `device_index`. Return NULL if there is none.

The topology `topology` does not necessarily have to match the current machine. For instance the topology may be an XML import of a remote host. I/O devices detection and the OpenCL component must be enabled in the topology.

**Note:**

The corresponding PCI device object can be obtained by looking at the OS device parent object (unless PCI devices are filtered out).

#### 22.34.2.4 static int hwloc_opencl_get_device_pci_busid (cl_device_id *device*, unsigned * *domain*, unsigned * *bus*, unsigned * *dev*, unsigned * *func*) [inline, static]

Return the domain, bus and device IDs of the OpenCL device `device`. Device `device` must match the local machine.

## 22.35 Interoperability with the CUDA Driver API

### Functions

- static int hwloc_cuda_get_device_pci_ids (hwloc_topology_t topology, CUdevice cudevice, int *domain, int *bus, int *dev)
- static int hwloc_cuda_get_device_cpuset (hwloc_topology_t topology, CUdevice cudevice, hwloc_cpuset_t set)
- static hwloc_obj_t hwloc_cuda_get_device_pcidev (hwloc_topology_t topology, CUdevice cudevice)
- static hwloc_obj_t hwloc_cuda_get_device_osdev (hwloc_topology_t topology, CUdevice cudevice)
- static hwloc_obj_t hwloc_cuda_get_device_osdev_by_index (hwloc_topology_t topology, unsigned idx)

### 22.35.1 Detailed Description

This interface offers ways to retrieve topology information about CUDA devices when using the CUDA Driver API.

### 22.35.2 Function Documentation

#### 22.35.2.1 static int hwloc_cuda_get_device_cpuset (hwloc_topology_t *topology*, CUdevice *cudevice*, hwloc_cpuset_t *set*) [inline, static]

Get the CPU set of logical processors that are physically close to device `cudevice`. Return the CPU set describing the locality of the CUDA device `cudevice`.

Topology `topology` and device `cudevice` must match the local machine. I/O devices detection and the CUDA component are not needed in the topology.

The function only returns the locality of the device. If more information about the device is needed, OS objects should be used instead, see hwloc_cuda_get_device_osdev() and hwloc_cuda_get_device_osdev_by_index().

This function is currently only implemented in a meaningful way for Linux; other systems will simply get a full cpuset.

#### 22.35.2.2 static hwloc_obj_t hwloc_cuda_get_device_osdev (hwloc_topology_t *topology*, CUdevice *cudevice*) [inline, static]

Get the hwloc OS device object corresponding to CUDA device `cudevice`. Return the hwloc OS device object that describes the given CUDA device `cudevice`. Return NULL if there is none.

Topology `topology` and device `cudevice` must match the local machine. I/O devices detection and the CUDA component must be enabled in the topology. If not, the locality of the object may still be found using hwloc_cuda_get_device_cpuset().

**Note:**

This function cannot work if PCI devices are filtered out.
The corresponding hwloc PCI device may be found by looking at the result parent pointer (unless PCI devices are filtered out).

#### 22.35.2.3 static hwloc_obj_t hwloc_cuda_get_device_osdev_by_index (hwloc_topology_t *topology*, unsigned *idx*) [inline, static]

Get the hwloc OS device object corresponding to the CUDA device whose index is `idx`. Return the OS device object describing the CUDA device whose index is `idx`. Return NULL if there is none.

The topology `topology` does not necessarily have to match the current machine. For instance the topology may be an XML import of a remote host. I/O devices detection and the CUDA component must be enabled in the topology.

**Note:**

The corresponding PCI device object can be obtained by looking at the OS device parent object (unless PCI devices are filtered out).
This function is identical to hwloc_cudart_get_device_osdev_by_index().

#### 22.35.2.4 static int hwloc_cuda_get_device_pci_ids (hwloc_topology_t *topology*, CUdevice *cudevice*, int * *domain*, int * *bus*, int * *dev*) [inline, static]

Return the domain, bus and device IDs of the CUDA device `cudevice`. Device `cudevice` must match the local machine.

#### 22.35.2.5 static hwloc_obj_t hwloc_cuda_get_device_pcidev (hwloc_topology_t *topology*, CUdevice *cudevice*) [inline, static]

Get the hwloc PCI device object corresponding to the CUDA device `cudevice`. Return the PCI device object describing the CUDA device `cudevice`. Return NULL if there is none.

Topology `topology` and device `cudevice` must match the local machine. I/O devices detection must be enabled in topology `topology`. The CUDA component is not needed in the topology.

# 22.36 Interoperability with the CUDA Runtime API

## Functions

- static int hwloc_cudart_get_device_pci_ids (hwloc_topology_t topology, int idx, int *domain, int *bus, int *dev)
- static int hwloc_cudart_get_device_cpuset (hwloc_topology_t topology, int idx, hwloc_cpuset_t set)
- static hwloc_obj_t hwloc_cudart_get_device_pcidev (hwloc_topology_t topology, int idx)
- static hwloc_obj_t hwloc_cudart_get_device_osdev_by_index (hwloc_topology_t topology, unsigned idx)

## 22.36.1 Detailed Description

This interface offers ways to retrieve topology information about CUDA devices when using the CUDA Runtime API.

## 22.36.2 Function Documentation

### 22.36.2.1 static int hwloc_cudart_get_device_cpuset (hwloc_topology_t *topology*, int *idx*, hwloc_cpuset_t *set*) [inline, static]

Get the CPU set of logical processors that are physically close to device `idx`. Return the CPU set describing the locality of the CUDA device whose index is `idx`.

Topology `topology` and device `idx` must match the local machine. I/O devices detection and the CUDA component are not needed in the topology.

The function only returns the locality of the device. If more information about the device is needed, OS objects should be used instead, see hwloc_cudart_get_device_osdev_by_index().

This function is currently only implemented in a meaningful way for Linux; other systems will simply get a full cpuset.

### 22.36.2.2 static hwloc_obj_t hwloc_cudart_get_device_osdev_by_index (hwloc_topology_t *topology*, unsigned *idx*) [inline, static]

Get the hwloc OS device object corresponding to the CUDA device whose index is `idx`. Return the OS device object describing the CUDA device whose index is `idx`. Return NULL if there is none.

The topology `topology` does not necessarily have to match the current machine. For instance the topology may be an XML import of a remote host. I/O devices detection and the CUDA component must be enabled in the topology. If not, the locality of the object may still be found using hwloc_cudart_get_device_cpuset().

**Note:**

The corresponding PCI device object can be obtained by looking at the OS device parent object (unless PCI devices are filtered out).
This function is identical to hwloc_cuda_get_device_osdev_by_index().

#### 22.36.2.3 static int hwloc_cudart_get_device_pci_ids (hwloc_topology_t *topology*, int *idx*, int * *domain*, int * *bus*, int * *dev*) [inline, static]

Return the domain, bus and device IDs of the CUDA device whose index is `idx`. Device index `idx` must match the local machine.

#### 22.36.2.4 static hwloc_obj_t hwloc_cudart_get_device_pcidev (hwloc_topology_t *topology*, int *idx*) [inline, static]

Get the hwloc PCI device object corresponding to the CUDA device whose index is `idx`. Return the PCI device object describing the CUDA device whose index is `idx`. Return NULL if there is none.

Topology `topology` and device `idx` must match the local machine. I/O devices detection must be enabled in topology `topology`. The CUDA component is not needed in the topology.

# 22.37 Interoperability with the NVIDIA Management Library

## Functions

- static int hwloc_nvml_get_device_cpuset (hwloc_topology_t topology, nvmlDevice_t device, hwloc_cpuset_t set)
- static hwloc_obj_t hwloc_nvml_get_device_osdev_by_index (hwloc_topology_t topology, unsigned idx)
- static hwloc_obj_t hwloc_nvml_get_device_osdev (hwloc_topology_t topology, nvmlDevice_t device)

## 22.37.1 Detailed Description

This interface offers ways to retrieve topology information about devices managed by the NVIDIA Management Library (NVML).

## 22.37.2 Function Documentation

### 22.37.2.1 static int hwloc_nvml_get_device_cpuset (hwloc_topology_t *topology*, nvmlDevice_t *device*, hwloc_cpuset_t *set*) [inline, static]

Get the CPU set of logical processors that are physically close to NVML device `device`. Return the CPU set describing the locality of the NVML device `device`.

Topology `topology` and device `device` must match the local machine. I/O devices detection and the NVML component are not needed in the topology.

The function only returns the locality of the device. If more information about the device is needed, OS objects should be used instead, see hwloc_nvml_get_device_osdev() and hwloc_nvml_get_device_osdev_by_index().

This function is currently only implemented in a meaningful way for Linux; other systems will simply get a full cpuset.

### 22.37.2.2 static hwloc_obj_t hwloc_nvml_get_device_osdev (hwloc_topology_t *topology*, nvmlDevice_t *device*) [inline, static]

Get the hwloc OS device object corresponding to NVML device `device`. Return the hwloc OS device object that describes the given NVML device `device`. Return NULL if there is none.

Topology `topology` and device `device` must match the local machine. I/O devices detection and the NVML component must be enabled in the topology. If not, the locality of the object may still be found using hwloc_nvml_get_device_cpuset().

**Note:**

The corresponding hwloc PCI device may be found by looking at the result parent pointer (unless PCI devices are filtered out).

#### 22.37.2.3 static hwloc_obj_t hwloc_nvml_get_device_osdev_by_index (hwloc_topology_t *topology*, unsigned *idx*) [inline, static]

Get the hwloc OS device object corresponding to the NVML device whose index is `idx`. Return the OS device object describing the NVML device whose index is `idx`. Returns NULL if there is none.

The topology `topology` does not necessarily have to match the current machine. For instance the topology may be an XML import of a remote host. I/O devices detection and the NVML component must be enabled in the topology.

**Note:**

The corresponding PCI device object can be obtained by looking at the OS device parent object (unless PCI devices are filtered out).

# 22.38 Interoperability with OpenGL displays

## Functions

- static hwloc_obj_t hwloc_gl_get_display_osdev_by_port_device (hwloc_topology_t topology, unsigned port, unsigned device)
- static hwloc_obj_t hwloc_gl_get_display_osdev_by_name (hwloc_topology_t topology, const char *name)
- static int hwloc_gl_get_display_by_osdev (hwloc_topology_t topology, hwloc_obj_t osdev, unsigned *port, unsigned *device)

## 22.38.1 Detailed Description

This interface offers ways to retrieve topology information about OpenGL displays.

Only the NVIDIA display locality information is currently available, using the NV-CONTROL X11 extension and the NVCtrl library.

## 22.38.2 Function Documentation

### 22.38.2.1 static int hwloc_gl_get_display_by_osdev (hwloc_topology_t *topology*, hwloc_obj_t *osdev*, unsigned * *port*, unsigned * *device*) [inline, static]

Get the OpenGL display port and device corresponding to the given hwloc OS object. Return the OpenGL display port (server) in `port` and device (screen) in `screen` that correspond to the given hwloc OS device object. Return `-1` if there is none.

The topology `topology` does not necessarily have to match the current machine. For instance the topology may be an XML import of a remote host. I/O devices detection and the GL component must be enabled in the topology.

### 22.38.2.2 static hwloc_obj_t hwloc_gl_get_display_osdev_by_name (hwloc_topology_t *topology*, const char * *name*) [inline, static]

Get the hwloc OS device object corresponding to the OpenGL display given by name. Return the OS device object describing the OpenGL display whose name is `name`, built as ":port.device" such as ":0.0" . Return NULL if there is none.

The topology `topology` does not necessarily have to match the current machine. For instance the topology may be an XML import of a remote host. I/O devices detection and the GL component must be enabled in the topology.

**Note:**

The corresponding PCI device object can be obtained by looking at the OS device parent object (unless PCI devices are filtered out).

### 22.38.2.3 static hwloc_obj_t hwloc_gl_get_display_osdev_by_port_device (hwloc_topology_t *topology*, unsigned *port*, unsigned *device*) [inline, static]

Get the hwloc OS device object corresponding to the OpenGL display given by port and device index. Return the OS device object describing the OpenGL display whose port (server) is `port` and device (screen)

is `device`. Return NULL if there is none.

The topology `topology` does not necessarily have to match the current machine. For instance the topology may be an XML import of a remote host. I/O devices detection and the GL component must be enabled in the topology.

**Note:**

The corresponding PCI device object can be obtained by looking at the OS device parent object (unless PCI devices are filtered out).

## 22.39 Interoperability with OpenFabrics

### Functions

- static int hwloc_ibv_get_device_cpuset (hwloc_topology_t topology, struct ibv_device *ibdev, hwloc_cpuset_t set)
- static hwloc_obj_t hwloc_ibv_get_device_osdev_by_name (hwloc_topology_t topology, const char *ibname)
- static hwloc_obj_t hwloc_ibv_get_device_osdev (hwloc_topology_t topology, struct ibv_device *ibdev)

### 22.39.1 Detailed Description

This interface offers ways to retrieve topology information about OpenFabrics devices (InfiniBand, Omni-Path, usNIC, etc).

### 22.39.2 Function Documentation

#### 22.39.2.1 static int hwloc_ibv_get_device_cpuset (hwloc_topology_t *topology*, struct ibv_device * *ibdev*, hwloc_cpuset_t *set*) [inline, static]

Get the CPU set of logical processors that are physically close to device `ibdev`. Return the CPU set describing the locality of the OpenFabrics device `ibdev` (InfiniBand, etc).

Topology `topology` and device `ibdev` must match the local machine. I/O devices detection is not needed in the topology.

The function only returns the locality of the device. If more information about the device is needed, OS objects should be used instead, see hwloc_ibv_get_device_osdev() and hwloc_ibv_get_device_osdev_by_name().

This function is currently only implemented in a meaningful way for Linux; other systems will simply get a full cpuset.

#### 22.39.2.2 static hwloc_obj_t hwloc_ibv_get_device_osdev (hwloc_topology_t *topology*, struct ibv_device * *ibdev*) [inline, static]

Get the hwloc OS device object corresponding to the OpenFabrics device `ibdev`. Return the OS device object describing the OpenFabrics device `ibdev` (InfiniBand, etc). Returns NULL if there is none.

Topology `topology` and device `ibdev` must match the local machine. I/O devices detection must be enabled in the topology. If not, the locality of the object may still be found using hwloc_ibv_get_device_cpuset().

**Note:**

The corresponding PCI device object can be obtained by looking at the OS device parent object.

#### 22.39.2.3 static hwloc_obj_t hwloc_ibv_get_device_osdev_by_name (hwloc_topology_t *topology*, const char * *ibname*) [inline, static]

Get the hwloc OS device object corresponding to the OpenFabrics device named `ibname`. Return the OS device object describing the OpenFabrics device (InfiniBand, Omni-Path, usNIC, etc) whose name is

`ibname` (mlx5_0, hfi1_0, usnic_0, qib0, etc). Returns NULL if there is none. The name `ibname` is usually obtained from ibv_get_device_name().

The topology `topology` does not necessarily have to match the current machine. For instance the topology may be an XML import of a remote host. I/O devices detection must be enabled in the topology.

**Note:**

The corresponding PCI device object can be obtained by looking at the OS device parent object.

## 22.40 Topology differences

### Data Structures

- union hwloc_topology_diff_obj_attr_u

  *One object attribute difference.*

- union hwloc_topology_diff_u

  *One element of a difference list between two topologies.*

### Typedefs

- typedef enum hwloc_topology_diff_obj_attr_type_e hwloc_topology_diff_obj_attr_type_t
- typedef enum hwloc_topology_diff_type_e hwloc_topology_diff_type_t
- typedef union hwloc_topology_diff_u * hwloc_topology_diff_t

### Enumerations

- enum hwloc_topology_diff_obj_attr_type_e { HWLOC_TOPOLOGY_DIFF_OBJ_ATTR_SIZE, HWLOC_TOPOLOGY_DIFF_OBJ_ATTR_NAME, HWLOC_TOPOLOGY_DIFF_OBJ_ATTR_INFO }
- enum hwloc_topology_diff_type_e { HWLOC_TOPOLOGY_DIFF_OBJ_ATTR, HWLOC_TOPOLOGY_DIFF_TOO_COMPLEX }
- enum hwloc_topology_diff_apply_flags_e { HWLOC_TOPOLOGY_DIFF_APPLY_REVERSE }

### Functions

- int hwloc_topology_diff_build (hwloc_topology_t topology, hwloc_topology_t newtopology, unsigned long flags, hwloc_topology_diff_t *diff)
- int hwloc_topology_diff_apply (hwloc_topology_t topology, hwloc_topology_diff_t diff, unsigned long flags)
- int hwloc_topology_diff_destroy (hwloc_topology_diff_t diff)
- int hwloc_topology_diff_load_xml (const char *xmlpath, hwloc_topology_diff_t *diff, char **refname)
- int hwloc_topology_diff_export_xml (hwloc_topology_diff_t diff, const char *refname, const char *xmlpath)
- int hwloc_topology_diff_load_xmlbuffer (const char *xmlbuffer, int buflen, hwloc_topology_diff_t *diff, char **refname)
- int hwloc_topology_diff_export_xmlbuffer (hwloc_topology_diff_t diff, const char *refname, char **xmlbuffer, int *buflen)

### 22.40.1 Detailed Description

Applications that manipulate many similar topologies, for instance one for each node of a homogeneous cluster, may want to compress topologies to reduce the memory footprint.

This file offers a way to manipulate the difference between topologies and export/import it to/from XML. Compression may therefore be achieved by storing one topology entirely while the others are only described by their differences with the former. The actual topology can be reconstructed when actually needed by applying the precomputed difference to the reference topology.

This interface targets very similar nodes. Only very simple differences between topologies are actually supported, for instance a change in the memory size, the name of the object, or some info attribute. More complex differences such as adding or removing objects cannot be represented in the difference structures and therefore return errors. Differences between object sets or topology-wide allowed sets, cannot be represented either.

It means that there is no need to apply the difference when looking at the tree organization (how many levels, how many objects per level, what kind of objects, CPU and node sets, etc) and when binding to objects. However the difference must be applied when looking at object attributes such as the name, the memory size or info attributes.

## 22.40.2 Typedef Documentation

### 22.40.2.1 typedef enum hwloc_topology_diff_obj_attr_type_e hwloc_topology_diff_obj_attr_type_t

Type of one object attribute difference.

### 22.40.2.2 typedef union hwloc_topology_diff_u * hwloc_topology_diff_t

One element of a difference list between two topologies.

### 22.40.2.3 typedef enum hwloc_topology_diff_type_e hwloc_topology_diff_type_t

Type of one element of a difference list.

## 22.40.3 Enumeration Type Documentation

### 22.40.3.1 enum hwloc_topology_diff_apply_flags_e

Flags to be given to hwloc_topology_diff_apply().

**Enumerator:**

***HWLOC_TOPOLOGY_DIFF_APPLY_REVERSE*** Apply topology diff in reverse direction.

### 22.40.3.2 enum hwloc_topology_diff_obj_attr_type_e

Type of one object attribute difference.

**Enumerator:**

***HWLOC_TOPOLOGY_DIFF_OBJ_ATTR_SIZE*** The object local memory is modified. The union is a hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_uint64_s (and the index field is ignored).

***HWLOC_TOPOLOGY_DIFF_OBJ_ATTR_NAME*** The object name is modified. The union is a hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_string_s (and the name field is ignored).

***HWLOC_TOPOLOGY_DIFF_OBJ_ATTR_INFO*** the value of an info attribute is modified. The union is a hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_string_s.

#### 22.40.3.3 enum hwloc_topology_diff_type_e

Type of one element of a difference list.

**Enumerator:**

***HWLOC_TOPOLOGY_DIFF_OBJ_ATTR*** An object attribute was changed. The union is a hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_s.

***HWLOC_TOPOLOGY_DIFF_TOO_COMPLEX*** The difference is too complex, it cannot be represented. The difference below this object has not been checked. hwloc_topology_diff_build() will return 1. The union is a hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_too_complex_s.

### 22.40.4 Function Documentation

#### 22.40.4.1 int hwloc_topology_diff_apply (hwloc_topology_t *topology*, hwloc_topology_diff_t *diff*, unsigned long *flags*)

Apply a topology diff to an existing topology. `flags` is an OR'ed set of hwloc_topology_diff_apply_flags_e.

The new topology is modified in place. hwloc_topology_dup() may be used to duplicate it before patching.

If the difference cannot be applied entirely, all previous applied elements are unapplied before returning.

**Returns:**

0 on success.
-N if applying the difference failed while trying to apply the N-th part of the difference. For instance -1 is returned if the very first difference element could not be applied.

#### 22.40.4.2 int hwloc_topology_diff_build (hwloc_topology_t *topology*, hwloc_topology_t *newtopology*, unsigned long *flags*, hwloc_topology_diff_t * *diff*)

Compute the difference between 2 topologies. The difference is stored as a list of hwloc_topology_diff_t entries starting at `diff`. It is computed by doing a depth-first traversal of both topology trees simultaneously.

If the difference between 2 objects is too complex to be represented (for instance if some objects have different types, or different numbers of children), a special diff entry of type HWLOC_TOPOLOGY_DIFF_TOO_COMPLEX is queued. The computation of the diff does not continue below these objects. So each such diff entry means that the difference between two subtrees could not be computed.

**Returns:**

0 if the difference can be represented properly.
0 with `diff` pointing to NULL if there is no difference between the topologies.

1 if the difference is too complex (see above). Some entries in the list will be of type HWLOC_-TOPOLOGY_DIFF_TOO_COMPLEX.
-1 on any other error.

**Note:**

`flags` is currently not used. It should be 0.
The output diff has to be freed with hwloc_topology_diff_destroy().
The output diff can only be exported to XML or passed to hwloc_topology_diff_apply() if 0 was returned, i.e. if no entry of type HWLOC_TOPOLOGY_DIFF_TOO_COMPLEX is listed.
The output diff may be modified by removing some entries from the list. The removed entries should be freed by passing them to to hwloc_topology_diff_destroy() (possible as another list).

#### 22.40.4.3 int hwloc_topology_diff_destroy (hwloc_topology_diff_t *diff*)

Destroy a list of topology differences.

#### 22.40.4.4 int hwloc_topology_diff_export_xml (hwloc_topology_diff_t *diff*, const char * *refname*, const char * *xmlpath*)

Export a list of topology differences to a XML file. If not `NULL`, `refname` defines an identifier string for the reference topology which was used as a base when computing this difference. This identifier is usually the name of the other XML file that contains the reference topology. This attribute is given back when reading the diff from XML.

#### 22.40.4.5 int hwloc_topology_diff_export_xmlbuffer (hwloc_topology_diff_t *diff*, const char * *refname*, char ** *xmlbuffer*, int * *buflen*)

Export a list of topology differences to a XML buffer. If not `NULL`, `refname` defines an identifier string for the reference topology which was used as a base when computing this difference. This identifier is usually the name of the other XML file that contains the reference topology. This attribute is given back when reading the diff from XML.

The returned buffer ends with a that is included in the returned length.

**Note:**

The XML buffer should later be freed with hwloc_free_xmlbuffer().

#### 22.40.4.6 int hwloc_topology_diff_load_xml (const char * *xmlpath*, hwloc_topology_diff_t * *diff*, char ** *refname*)

Load a list of topology differences from a XML file. If not `NULL`, `refname` will be filled with the identifier string of the reference topology for the difference file, if any was specified in the XML file. This identifier is usually the name of the other XML file that contains the reference topology.

**Note:**

the pointer returned in refname should later be freed by the caller.

#### 22.40.4.7 int hwloc_topology_diff_load_xmlbuffer (const char * *xmlbuffer*, int *buflen*, hwloc_topology_diff_t * *diff*, char ** *refname*)

Load a list of topology differences from a XML buffer. If not `NULL`, `refname` will be filled with the identifier string of the reference topology for the difference file, if any was specified in the XML file. This identifier is usually the name of the other XML file that contains the reference topology.

**Note:**

the pointer returned in refname should later be freed by the caller.

# 22.41 Sharing topologies between processes

## Functions

- int hwloc_shmem_topology_get_length (hwloc_topology_t topology, size_t *lengthp, unsigned long flags)
- int hwloc_shmem_topology_write (hwloc_topology_t topology, int fd, hwloc_uint64_t fileoffset, void *mmap_address, size_t length, unsigned long flags)
- int hwloc_shmem_topology_adopt (hwloc_topology_t *topologyp, int fd, hwloc_uint64_t fileoffset, void *mmap_address, size_t length, unsigned long flags)

### 22.41.1 Detailed Description

These functions are used to share a topology between processes by duplicating it into a file-backed shared-memory buffer.

The master process must first get the required shared-memory size for storing this topology with hwloc_shmem_topology_get_length().

Then it must find a virtual memory area of that size that is available in all processes (identical virtual addresses in all processes). On Linux, this can be done by comparing holes found in /proc/<pid>/maps for each process.

Once found, it must open a destination file for storing the buffer, and pass it to hwloc_shmem_topology_write() together with virtual memory address and length obtained above.

Other processes may then adopt this shared topology by opening the same file and passing it to hwloc_shmem_topology_adopt() with the exact same virtual memory address and length.

### 22.41.2 Function Documentation

#### 22.41.2.1 int hwloc_shmem_topology_adopt (hwloc_topology_t * *topologyp*, int *fd*, hwloc_uint64_t *fileoffset*, void * *mmap_address*, size_t *length*, unsigned long *flags*)

Adopt a shared memory topology stored in a file. Map a file in virtual memory and adopt the topology that was previously stored there with hwloc_shmem_topology_write().

The returned adopted topology in `topologyp` can be used just like any topology. And it must be destroyed with hwloc_topology_destroy() as usual.

However the topology is read-only. For instance, it cannot be modified with hwloc_topology_restrict() and object userdata pointers cannot be changed.

The segment of the file pointed by descriptor `fd`, starting at offset `fileoffset`, and of length `length` (in bytes), will be mapped at virtual address `mmap_address`.

The file pointed by descriptor `fd`, the offset `fileoffset`, the requested mapping virtual address `mmap_address` and the length `length` must be identical to what was given to hwloc_shmem_topology_write() earlier.

**Note:**

Flags `flags` are currently unused, must be 0.
The object userdata pointer should not be used unless the process that created the shared topology also placed userdata-pointed buffers in shared memory.
This function takes care of calling hwloc_topology_abi_check().

**Returns:**

-1 with errno set to EBUSY if the virtual memory mapping defined by `mmap_address` and `length` isn't available in the process.
-1 with errno set to EINVAL if `fileoffset`, `mmap_address` or `length` aren't page-aligned, or do not match what was given to hwloc_shmem_topology_write() earlier.
-1 with errno set to EINVAL if the layout of the topology structure is different between the writer process and the adopter process.

#### 22.41.2.2 int hwloc_shmem_topology_get_length (hwloc_topology_t *topology*, size_t * *lengthp*, unsigned long *flags*)

Get the required shared memory length for storing a topology. This length (in bytes) must be used in hwloc_shmem_topology_write() and hwloc_shmem_topology_adopt() later.

**Note:**

Flags `flags` are currently unused, must be 0.

#### 22.41.2.3 int hwloc_shmem_topology_write (hwloc_topology_t *topology*, int *fd*, hwloc_uint64_t *fileoffset*, void * *mmap_address*, size_t *length*, unsigned long *flags*)

Duplicate a topology to a shared memory file. Temporarily map a file in virtual memory and duplicate the topology `topology` by allocating duplicates in there.

The segment of the file pointed by descriptor `fd`, starting at offset `fileoffset`, and of length `length` (in bytes), will be temporarily mapped at virtual address `mmap_address` during the duplication.

The mapping length `length` must have been previously obtained with hwloc_shmem_topology_get_length() and the topology must not have been modified in the meantime.

**Note:**

Flags `flags` are currently unused, must be 0.
The object userdata pointer is duplicated but the pointed buffer is not. However the caller may also allocate it manually in shared memory to share it as well.

**Returns:**

-1 with errno set to EBUSY if the virtual memory mapping defined by `mmap_address` and `length` isn't available in the process.
-1 with errno set to EINVAL if `fileoffset`, `mmap_address` or `length` aren't page-aligned.

# 22.42 Components and Plugins: Discovery components

## Data Structures

- struct hwloc_disc_component

  *Discovery component structure.*

## 22.43 Components and Plugins: Discovery backends

### Data Structures

- struct hwloc_disc_status

  *Discovery status structure.*

- struct hwloc_backend

  *Discovery backend structure.*

### Typedefs

- typedef enum hwloc_disc_phase_e hwloc_disc_phase_t

### Enumerations

- enum hwloc_disc_phase_e {

  HWLOC_DISC_PHASE_GLOBAL, HWLOC_DISC_PHASE_CPU, HWLOC_DISC_PHASE_-MEMORY, HWLOC_DISC_PHASE_PCI,

  HWLOC_DISC_PHASE_IO, HWLOC_DISC_PHASE_MISC, HWLOC_DISC_PHASE_-ANNOTATE, HWLOC_DISC_PHASE_TWEAK }
- enum hwloc_disc_status_flag_e { HWLOC_DISC_STATUS_FLAG_GOT_ALLOWED_-RESOURCES }

### Functions

- struct hwloc_backend * hwloc_backend_alloc (struct hwloc_topology *topology, struct hwloc_-disc_component *component)
- int hwloc_backend_enable (struct hwloc_backend *backend)

### 22.43.1 Typedef Documentation

#### 22.43.1.1 typedef enum hwloc_disc_phase_e hwloc_disc_phase_t

Discovery phase.

### 22.43.2 Enumeration Type Documentation

#### 22.43.2.1 enum hwloc_disc_phase_e

Discovery phase.

**Enumerator:**

***HWLOC_DISC_PHASE_GLOBAL*** xml or synthetic, platform-specific components such as bgq. Discovers everything including CPU, memory, I/O and everything else. A component with a Global phase usually excludes all other phases.

***HWLOC_DISC_PHASE_CPU*** CPU discovery.

***HWLOC_DISC_PHASE_MEMORY*** Attach memory to existing CPU objects.

***HWLOC_DISC_PHASE_PCI*** Attach PCI devices and bridges to existing CPU objects.

***HWLOC_DISC_PHASE_IO*** I/O discovery that requires PCI devices (OS devices such as OpenCL, CUDA, etc.).

***HWLOC_DISC_PHASE_MISC*** Misc objects that gets added below anything else.

***HWLOC_DISC_PHASE_ANNOTATE*** Annotating existing objects, adding distances, etc.

***HWLOC_DISC_PHASE_TWEAK*** Final tweaks to a ready-to-use topology. This phase runs once the topology is loaded, before it is returned to the topology. Hence it may only use the main hwloc API for modifying the topology, for instance by restricting it, adding info attributes, etc.

#### 22.43.2.2 enum hwloc_disc_status_flag_e

Discovery status flags.

**Enumerator:**

***HWLOC_DISC_STATUS_FLAG_GOT_ALLOWED_RESOURCES*** The sets of allowed resources were already retrieved.

### 22.43.3 Function Documentation

#### 22.43.3.1 struct hwloc_backend* hwloc_backend_alloc (struct hwloc_topology * *topology*, struct hwloc_disc_component * *component*) [read]

Allocate a backend structure, set good default values, initialize backend->component and topology, etc. The caller will then modify whatever needed, and call hwloc_backend_enable().

#### 22.43.3.2 int hwloc_backend_enable (struct hwloc_backend * *backend*)

Enable a previously allocated and setup backend.

## 22.44 Components and Plugins: Generic components

### Data Structures

- struct hwloc_component

  *Generic component structure.*

### Typedefs

- typedef enum hwloc_component_type_e hwloc_component_type_t

### Enumerations

- enum hwloc_component_type_e { HWLOC_COMPONENT_TYPE_DISC, HWLOC_COMPONENT_TYPE_XML }

### 22.44.1 Typedef Documentation

#### 22.44.1.1 typedef enum hwloc_component_type_e hwloc_component_type_t

Generic component type.

### 22.44.2 Enumeration Type Documentation

#### 22.44.2.1 enum hwloc_component_type_e

Generic component type.

**Enumerator:**

***HWLOC_COMPONENT_TYPE_DISC*** The data field must point to a struct hwloc_disc_component.

***HWLOC_COMPONENT_TYPE_XML*** The data field must point to a struct hwloc_xml_component.

# 22.45 Components and Plugins: Core functions to be used by components

## Typedefs

- typedef void(∗ hwloc_report_error_t )(const char ∗msg, int line)

## Functions

- struct hwloc_obj ∗ hwloc_insert_object_by_cpuset (struct hwloc_topology ∗topology, hwloc_obj_t obj)
- void hwloc_report_os_error (const char ∗msg, int line)
- int hwloc_hide_errors (void)
- struct hwloc_obj ∗ hwloc__insert_object_by_cpuset (struct hwloc_topology ∗topology, hwloc_obj_t root, hwloc_obj_t obj, hwloc_report_error_t report_error)
- void hwloc_insert_object_by_parent (struct hwloc_topology ∗topology, hwloc_obj_t parent, hwloc_obj_t obj)
- hwloc_obj_t hwloc_alloc_setup_object (hwloc_topology_t topology, hwloc_obj_type_t type, unsigned os_index)
- int hwloc_obj_add_children_sets (hwloc_obj_t obj)
- int hwloc_topology_reconnect (hwloc_topology_t topology, unsigned long flags)
- static int hwloc_plugin_check_namespace (const char ∗pluginname, const char ∗symbol)

### 22.45.1 Typedef Documentation

#### 22.45.1.1 typedef void(∗ hwloc_report_error_t)(const char ∗msg, int line)

Type of error callbacks during object insertion.

### 22.45.2 Function Documentation

#### 22.45.2.1 struct hwloc_obj∗ hwloc__insert_object_by_cpuset (struct hwloc_topology ∗ *topology*, hwloc_obj_t *root*, hwloc_obj_t *obj*, hwloc_report_error_t *report_error*) [read]

Add an object to the topology and specify which error callback to use. This function is similar to hwloc_-insert_object_by_cpuset() but it allows specifying where to start insertion from (if `root` is NULL, the topology root object is used), and specifying the error callback.

#### 22.45.2.2 hwloc_obj_t hwloc_alloc_setup_object (hwloc_topology_t *topology*, hwloc_obj_type_t *type*, unsigned *os_index*)

Allocate and initialize an object of the given type and physical index. If `os_index` is unknown or irrelevant, use `HWLOC_UNKNOWN_INDEX`.

#### 22.45.2.3 int hwloc_hide_errors (void)

Check whether insertion errors are hidden.

#### 22.45.2.4 struct hwloc_obj* hwloc_insert_object_by_cpuset (struct hwloc_topology * *topology*, hwloc_obj_t *obj*) [read]

Add an object to the topology. It is sorted along the tree of other objects according to the inclusion of cpusets, to eventually be added as a child of the smallest object including this object.

If the cpuset is empty, the type of the object (and maybe some attributes) must be enough to find where to insert the object. This is especially true for NUMA nodes with memory and no CPUs.

The given object should not have children.

This shall only be called before levels are built.

In case of error, hwloc_report_os_error() is called.

The caller should check whether the object type is filtered-out before calling this function.

The topology cpuset/nodesets will be enlarged to include the object sets.

Returns the object on success. Returns NULL and frees obj on error. Returns another object and frees obj if it was merged with an identical pre-existing object.

#### 22.45.2.5 void hwloc_insert_object_by_parent (struct hwloc_topology * *topology*, hwloc_obj_t *parent*, hwloc_obj_t *obj*)

Insert an object somewhere in the topology. It is added as the last child of the given parent. The cpuset is completely ignored, so strange objects such as I/O devices should preferably be inserted with this.

When used for "normal" children with cpusets (when importing from XML when duplicating a topology), the caller should make sure that:

- children are inserted in order,
- children cpusets do not intersect.

The given object may have normal, I/O or Misc children, as long as they are in order as well. These children must have valid parent and next_sibling pointers.

The caller should check whether the object type is filtered-out before calling this function.

#### 22.45.2.6 int hwloc_obj_add_children_sets (hwloc_obj_t *obj*)

Setup object cpusets/nodesets by OR'ing its children. Used when adding an object late in the topology. Will update the new object by OR'ing all its new children sets.

Used when PCI backend adds a hostbridge parent, when distances add a new Group, etc.

#### 22.45.2.7 static int hwloc_plugin_check_namespace (const char * *pluginname*, const char * *symbol*) [inline, static]

Make sure that plugins can lookup core symbols. This is a sanity check to avoid lazy-lookup failures when libhwloc is loaded within a plugin, and later tries to load its own plugins. This may fail (and abort the program) if libhwloc symbols are in a private namespace.

**Returns:**

0 on success.

-1 if the plugin cannot be successfully loaded. The caller plugin init() callback should return a negative error code as well.

Plugins should call this function in their init() callback to avoid later crashes if lazy symbol resolution is used by the upper layer that loaded hwloc (e.g. OpenCL implementations using dlopen with RTLD_LAZY).

**Note:**

The build system must define HWLOC_INSIDE_PLUGIN if and only if building the caller as a plugin. This function should remain inline so plugins can call it even when they cannot find libhwloc symbols.

#### 22.45.2.8 void hwloc_report_os_error (const char * *msg*, int *line*)

Report an insertion error from a backend.

#### 22.45.2.9 int hwloc_topology_reconnect (hwloc_topology_t *topology*, unsigned long *flags*)

Request a reconnection of children and levels in the topology. May be used by backends during discovery if they need arrays or lists of object within levels or children to be fully connected.

`flags` is currently unused, must 0.

## 22.46 Components and Plugins: Filtering objects

### Functions

- static int hwloc_filter_check_pcidev_subtype_important (unsigned classid)
- static int hwloc_filter_check_osdev_subtype_important (hwloc_obj_osdev_type_t subtype)
- static int hwloc_filter_check_keep_object_type (hwloc_topology_t topology, hwloc_obj_type_t type)
- static int hwloc_filter_check_keep_object (hwloc_topology_t topology, hwloc_obj_t obj)

### 22.46.1 Function Documentation

#### 22.46.1.1 static int hwloc_filter_check_keep_object (hwloc_topology_t *topology*, hwloc_obj_t *obj*) [inline, static]

Check whether the given object should be filtered-out.

**Returns:**

1 if the object type should be kept, 0 otherwise.

#### 22.46.1.2 static int hwloc_filter_check_keep_object_type (hwloc_topology_t *topology*, hwloc_obj_type_t *type*) [inline, static]

Check whether a non-I/O object type should be filtered-out. Cannot be used for I/O objects.

**Returns:**

1 if the object type should be kept, 0 otherwise.

#### 22.46.1.3 static int hwloc_filter_check_osdev_subtype_important (hwloc_obj_osdev_type_t *subtype*) [inline, static]

Check whether the given OS device subtype is important.

**Returns:**

1 if important, 0 otherwise.

#### 22.46.1.4 static int hwloc_filter_check_pcidev_subtype_important (unsigned *classid*) [inline, static]

Check whether the given PCI device classid is important.

**Returns:**

1 if important, 0 otherwise.

# 22.47 Components and Plugins: helpers for PCI discovery

## Functions

- unsigned hwloc_pcidisc_find_cap (const unsigned char *config, unsigned cap)
- int hwloc_pcidisc_find_linkspeed (const unsigned char *config, unsigned offset, float *linkspeed)
- hwloc_obj_type_t hwloc_pcidisc_check_bridge_type (unsigned device_class, const unsigned char *config)
- int hwloc_pcidisc_find_bridge_buses (unsigned domain, unsigned bus, unsigned dev, unsigned func, unsigned *secondary_busp, unsigned *subordinate_busp, const unsigned char *config)
- void hwloc_pcidisc_tree_insert_by_busid (struct hwloc_obj **treep, struct hwloc_obj *obj)
- int hwloc_pcidisc_tree_attach (struct hwloc_topology *topology, struct hwloc_obj *tree)

## 22.47.1 Function Documentation

### 22.47.1.1 hwloc_obj_type_t hwloc_pcidisc_check_bridge_type (unsigned *device_class*, const unsigned char * *config*)

Return the hwloc object type (PCI device or Bridge) for the given class and configuration space. This function requires 16 bytes of common configuration header at the beginning of config.

### 22.47.1.2 int hwloc_pcidisc_find_bridge_buses (unsigned *domain*, unsigned *bus*, unsigned *dev*, unsigned *func*, unsigned * *secondary_busp*, unsigned * *subordinate_busp*, const unsigned char * *config*)

Fills the attributes of the given PCI bridge using the given PCI config space. This function requires 32 bytes of common configuration header at the beginning of config.

Returns -1 and destroys /p obj if bridge fields are invalid.

### 22.47.1.3 unsigned hwloc_pcidisc_find_cap (const unsigned char * *config*, unsigned *cap*)

Return the offset of the given capability in the PCI config space buffer. This function requires a 256-bytes config space. Unknown/unavailable bytes should be set to 0xff.

### 22.47.1.4 int hwloc_pcidisc_find_linkspeed (const unsigned char * *config*, unsigned *offset*, float * *linkspeed*)

Fill linkspeed by reading the PCI config space where PCI_CAP_ID_EXP is at position offset. Needs 20 bytes of EXP capability block starting at offset in the config space for registers up to link status.

### 22.47.1.5 int hwloc_pcidisc_tree_attach (struct hwloc_topology * *topology*, struct hwloc_obj * *tree*)

Add some hostbridges on top of the given tree of PCI objects and attach them to the topology. Other backends may lookup PCI objects or localities (for instance to attach OS devices) by using hwloc_pcidisc_-find_by_busid() or hwloc_pcidisc_find_busid_parent().

#### 22.47.1.6 void hwloc_pcidisc_tree_insert_by_busid (struct hwloc_obj ** *treep*, struct hwloc_obj * *obj*)

Insert a PCI object in the given PCI tree by looking at PCI bus IDs. If `treep` points to `NULL`, the new object is inserted there.

# 22.48 Components and Plugins: finding PCI objects during other discoveries

## Functions

- struct hwloc_obj * hwloc_pci_find_parent_by_busid (struct hwloc_topology *topology, unsigned domain, unsigned bus, unsigned dev, unsigned func)

## 22.48.1 Function Documentation

### 22.48.1.1 struct hwloc_obj* hwloc_pci_find_parent_by_busid (struct hwloc_topology * *topology*, unsigned *domain*, unsigned *bus*, unsigned *dev*, unsigned *func*) [read]

Find the normal parent of a PCI bus ID. Look at PCI affinity to find out where the given PCI bus ID should be attached.

This function should be used to attach an I/O device under the corresponding PCI object (if any), or under a normal (non-I/O) object with same locality.

# 22.49 Netloc API

## Enumerations

- enum {

  NETLOC_SUCCESS = 0, NETLOC_ERROR = -1, NETLOC_ERROR_NOTDIR = -2, NETLOC_-ERROR_NOENT = -3,

  NETLOC_ERROR_EMPTY = -4, NETLOC_ERROR_MULTIPLE = -5, NETLOC_ERROR_-NOT_IMPL = -6, NETLOC_ERROR_EXISTS = -7,

  NETLOC_ERROR_NOT_FOUND = -8, NETLOC_ERROR_MAX = -9 }

## 22.49.1 Enumeration Type Documentation

### 22.49.1.1 anonymous enum

Return codes

**Enumerator:**

***NETLOC_SUCCESS*** Success

***NETLOC_ERROR*** Error: General condition

***NETLOC_ERROR_NOTDIR*** Error: URI is not a directory

***NETLOC_ERROR_NOENT*** Error: URI is invalid, no such entry

***NETLOC_ERROR_EMPTY*** Error: No networks found

***NETLOC_ERROR_MULTIPLE*** Error: Multiple matching networks found

***NETLOC_ERROR_NOT_IMPL*** Error: Interface not implemented

***NETLOC_ERROR_EXISTS*** Error: If the entry already exists when trying to add to a lookup table

***NETLOC_ERROR_NOT_FOUND*** Error: No path found

***NETLOC_ERROR_MAX*** Error: Enum upper bound marker. No errors less than this number Will not be returned externally.

# Chapter 23

# Data Structure Documentation

## 23.1 hwloc_backend Struct Reference

Discovery backend structure.

`#include <plugins.h>`

### Data Fields

- unsigned phases
- unsigned long flags
- int is_thissystem
- void * private_data
- void(* disable )(struct hwloc_backend *backend)
- int(* discover )(struct hwloc_backend *backend, struct hwloc_disc_status *status)
- int(* get_pci_busid_cpuset )(struct hwloc_backend *backend, struct hwloc_pcidev_attr_s *busid, hwloc_bitmap_t cpuset)

### 23.1.1 Detailed Description

Discovery backend structure. A backend is the instantiation of a discovery component. When a component gets enabled for a topology, its instantiate() callback creates a backend.

hwloc_backend_alloc() initializes all fields to default values that the component may change (except "component" and "next") before enabling the backend with hwloc_backend_enable().

Most backends assume that the topology is_thissystem flag is set because they talk to the underlying operating system. However they may still be used in topologies without the is_thissystem flag for debugging reasons. In practice, they are usually auto-disabled in such cases (excluded by xml or synthetic backends, or by environment variables when changing the Linux fsroot or the x86 cpuid path).

### 23.1.2 Field Documentation

#### 23.1.2.1 void(* hwloc_backend::disable)(struct hwloc_backend *backend)

Callback for freeing the private_data. May be NULL.

#### 23.1.2.2 int(∗ hwloc_backend::discover)(struct hwloc_backend ∗backend, struct hwloc_disc_status ∗status)

Main discovery callback. returns -1 on error, either because it couldn't add its objects ot the existing topology, or because of an actual discovery/gathering failure. May be NULL.

#### 23.1.2.3 unsigned long hwloc_backend::flags

Backend flags, currently always 0.

#### 23.1.2.4 int(∗ hwloc_backend::get_pci_busid_cpuset)(struct hwloc_backend ∗backend, struct hwloc_pcidev_attr_s ∗busid, hwloc_bitmap_t cpuset)

Callback to retrieve the locality of a PCI object. Called by the PCI core when attaching PCI hierarchy to CPU objects. May be NULL.

#### 23.1.2.5 int hwloc_backend::is_thissystem

Backend-specific 'is_thissystem' property. Set to 0 if the backend disables the thissystem flag for this topology (e.g. loading from xml or synthetic string, or using a different fsroot on Linux, or a x86 CPUID dump). Set to -1 if the backend doesn't care (default).

#### 23.1.2.6 unsigned hwloc_backend::phases

Discovery phases performed by this component, possibly without some of them if excluded by other components. OR'ed set of hwloc_disc_phase_t.

#### 23.1.2.7 void∗ hwloc_backend::private_data

Backend private data, or NULL if none.

The documentation for this struct was generated from the following file:

- plugins.h

## 23.2 hwloc_obj_attr_u::hwloc_bridge_attr_s Struct Reference

Bridge specific Object Attribues.

`#include <hwloc.h>`

### Data Fields

- union {
  struct hwloc_pcidev_attr_s pci
  } upstream

- hwloc_obj_bridge_type_t upstream_type
- union {
  struct {
  unsigned short domain
  unsigned char secondary_bus
  unsigned char subordinate_bus
  } pci
  } downstream

- hwloc_obj_bridge_type_t downstream_type
- unsigned depth

### 23.2.1 Detailed Description

Bridge specific Object Attribues.

### 23.2.2 Field Documentation

#### 23.2.2.1 unsigned hwloc_obj_attr_u::hwloc_bridge_attr_s::depth

#### 23.2.2.2 unsigned short hwloc_obj_attr_u::hwloc_bridge_attr_s::domain

#### 23.2.2.3 union { ... } hwloc_obj_attr_u::hwloc_bridge_attr_s::downstream

#### 23.2.2.4 hwloc_obj_bridge_type_t hwloc_obj_attr_u::hwloc_bridge_attr_s::downstream_type

#### 23.2.2.5 struct { ... } hwloc_obj_attr_u::hwloc_bridge_attr_s::pci

#### 23.2.2.6 struct hwloc_pcidev_attr_s hwloc_obj_attr_u::hwloc_bridge_attr_s::pci [read]

#### 23.2.2.7 unsigned char hwloc_obj_attr_u::hwloc_bridge_attr_s::secondary_bus

#### 23.2.2.8 unsigned char hwloc_obj_attr_u::hwloc_bridge_attr_s::subordinate_bus

#### 23.2.2.9 union { ... } hwloc_obj_attr_u::hwloc_bridge_attr_s::upstream

#### 23.2.2.10 hwloc_obj_bridge_type_t hwloc_obj_attr_u::hwloc_bridge_attr_s::upstream_type

The documentation for this struct was generated from the following file:

- hwloc.h

## 23.3 hwloc_obj_attr_u::hwloc_cache_attr_s Struct Reference

Cache-specific Object Attributes.

`#include <hwloc.h>`

### Data Fields

- hwloc_uint64_t size
- unsigned depth
- unsigned linesize
- int associativity
- hwloc_obj_cache_type_t type

### 23.3.1 Detailed Description

Cache-specific Object Attributes.

### 23.3.2 Field Documentation

#### 23.3.2.1 int hwloc_obj_attr_u::hwloc_cache_attr_s::associativity

Ways of associativity, -1 if fully associative, 0 if unknown.

#### 23.3.2.2 unsigned hwloc_obj_attr_u::hwloc_cache_attr_s::depth

Depth of cache (e.g., L1, L2, ...etc.).

#### 23.3.2.3 unsigned hwloc_obj_attr_u::hwloc_cache_attr_s::linesize

Cache-line size in bytes. 0 if unknown.

#### 23.3.2.4 hwloc_uint64_t hwloc_obj_attr_u::hwloc_cache_attr_s::size

Size of cache in bytes.

#### 23.3.2.5 hwloc_obj_cache_type_t hwloc_obj_attr_u::hwloc_cache_attr_s::type

Cache type.

The documentation for this struct was generated from the following file:

- hwloc.h

## 23.4 hwloc_cl_device_topology_amd Union Reference

`#include <opencl.h>`

### Data Fields

- struct {
  cl_uint type
  cl_uint data [5]
  } raw

- struct {
  cl_uint type
  cl_char unused [17]
  cl_char bus
  cl_char device
  cl_char function
  } pcie

### 23.4.1 Field Documentation

#### 23.4.1.1 cl_char hwloc_cl_device_topology_amd::bus

#### 23.4.1.2 cl_uint hwloc_cl_device_topology_amd::data[5]

#### 23.4.1.3 cl_char hwloc_cl_device_topology_amd::device

#### 23.4.1.4 cl_char hwloc_cl_device_topology_amd::function

#### 23.4.1.5 struct { ... } hwloc_cl_device_topology_amd::pcie

#### 23.4.1.6 struct { ... } hwloc_cl_device_topology_amd::raw

#### 23.4.1.7 cl_uint hwloc_cl_device_topology_amd::type

#### 23.4.1.8 cl_char hwloc_cl_device_topology_amd::unused[17]

The documentation for this union was generated from the following file:

- opencl.h

## 23.5 hwloc_component Struct Reference

Generic component structure.

`#include <plugins.h>`

### Data Fields

- unsigned abi
- int(∗ init )(unsigned long flags)
- void(∗ finalize )(unsigned long flags)
- hwloc_component_type_t type
- unsigned long flags
- void ∗ data

### 23.5.1 Detailed Description

Generic component structure. Generic components structure, either statically listed by configure in static-components.h or dynamically loaded as a plugin.

### 23.5.2 Field Documentation

#### 23.5.2.1 unsigned hwloc_component::abi

Component ABI version, set to HWLOC_COMPONENT_ABI.

#### 23.5.2.2 void∗ hwloc_component::data

Component data, pointing to a struct hwloc_disc_component or struct hwloc_xml_component.

#### 23.5.2.3 void(∗ hwloc_component::finalize)(unsigned long flags)

Process-wide component termination callback. This optional callback is called after unregistering the component from the hwloc core (before unloading the plugin).

`flags` is always 0 for now.

**Note:**

If the component uses ltdl for loading its own plugins, it should load/unload them only in init() and finalize(), to avoid race conditions with hwloc's use of ltdl.

#### 23.5.2.4 unsigned long hwloc_component::flags

Component flags, unused for now.

#### 23.5.2.5 int(* hwloc_component::init)(unsigned long flags)

Process-wide component initialization callback. This optional callback is called when the component is registered to the hwloc core (after loading the plugin).

When the component is built as a plugin, this callback should call hwloc_check_plugin_namespace() and return an negative error code on error.

`flags` is always 0 for now.

**Returns:**

0 on success, or a negative code on error.

**Note:**

If the component uses ltdl for loading its own plugins, it should load/unload them only in init() and finalize(), to avoid race conditions with hwloc's use of ltdl.

#### 23.5.2.6 hwloc_component_type_t hwloc_component::type

Component type.

The documentation for this struct was generated from the following file:

- plugins.h

## 23.6 hwloc_disc_component Struct Reference

Discovery component structure.

`#include <plugins.h>`

### Data Fields

- const char * name
- unsigned phases
- unsigned excluded_phases
- struct hwloc_backend *(* instantiate )(struct hwloc_topology *topology, struct hwloc_disc_component *component, unsigned excluded_phases, const void *data1, const void *data2, const void *data3)
- unsigned priority
- unsigned enabled_by_default

### 23.6.1 Detailed Description

Discovery component structure. This is the major kind of components, taking care of the discovery. They are registered by generic components, either statically-built or as plugins.

### 23.6.2 Field Documentation

#### 23.6.2.1 unsigned hwloc_disc_component::enabled_by_default

Enabled by default. If unset, if will be disabled unless explicitly requested.

#### 23.6.2.2 unsigned hwloc_disc_component::excluded_phases

Component phases to exclude, as an OR'ed set of hwloc_disc_phase_t. For a GLOBAL component, this usually includes all other phases (~UL).

Other components only exclude types that may bring conflicting topology information. MISC components should likely not be excluded since they usually bring non-primary additional information.

#### 23.6.2.3 struct hwloc_backend*(* hwloc_disc_component::instantiate)(struct hwloc_topology *topology, struct hwloc_disc_component *component, unsigned excluded_phases, const void *data1, const void *data2, const void *data3) [read]

Instantiate callback to create a backend from the component. Parameters data1, data2, data3 are NULL except for components that have special enabling routines such as hwloc_topology_set_xml().

#### 23.6.2.4 const char* hwloc_disc_component::name

Name. If this component is built as a plugin, this name does not have to match the plugin filename.

#### 23.6.2.5 unsigned hwloc_disc_component::phases

Discovery phases performed by this component. OR'ed set of hwloc_disc_phase_t.

#### 23.6.2.6 unsigned hwloc_disc_component::priority

Component priority. Used to sort topology->components, higher priority first. Also used to decide between two components with the same name. Usual values are 50 for native OS (or platform) components, 45 for x86, 40 for no-OS fallback, 30 for global components (xml, synthetic), 20 for pci, 10 for other misc components (opencl etc.).

The documentation for this struct was generated from the following file:

- plugins.h

## 23.7 hwloc_disc_status Struct Reference

Discovery status structure.

`#include <plugins.h>`

### Data Fields

- hwloc_disc_phase_t phase
- unsigned excluded_phases
- unsigned long flags

### 23.7.1 Detailed Description

Discovery status structure. Used by the core and backends to inform about what has been/is being done during the discovery process.

### 23.7.2 Field Documentation

#### 23.7.2.1 unsigned hwloc_disc_status::excluded_phases

Dynamically excluded phases. If a component decides during discovery that some phases are no longer needed.

#### 23.7.2.2 unsigned long hwloc_disc_status::flags

OR'ed set of hwloc_disc_status_flag_e.

#### 23.7.2.3 hwloc_disc_phase_t hwloc_disc_status::phase

The current discovery phase that is performed. Must match one of the phases in the component phases field.

The documentation for this struct was generated from the following file:

- plugins.h

## 23.8 hwloc_distances_s Struct Reference

Matrix of distances between a set of objects.

`#include <distances.h>`

### Data Fields

- unsigned nbobjs
- hwloc_obj_t * objs
- unsigned long kind
- hwloc_uint64_t * values

### 23.8.1 Detailed Description

Matrix of distances between a set of objects. This matrix often contains latencies between NUMA nodes (as reported in the System Locality Distance Information Table (SLIT) in the ACPI specification), which may or may not be physically accurate. It corresponds to the latency for accessing the memory of one node from a core in another node. The corresponding kind is HWLOC_DISTANCES_KIND_FROM_OS | HWLOC_DISTANCES_KIND_FROM_USER.

The matrix may also contain bandwidths between random sets of objects, possibly provided by the user, as specified in the `kind` attribute.

### 23.8.2 Field Documentation

#### 23.8.2.1 unsigned long hwloc_distances_s::kind

OR'ed set of hwloc_distances_kind_e.

#### 23.8.2.2 unsigned hwloc_distances_s::nbobjs

Number of objects described by the distance matrix.

#### 23.8.2.3 hwloc_obj_t* hwloc_distances_s::objs

Array of objects described by the distance matrix. These objects are not in any particular order, see hwloc_distances_obj_index() and hwloc_distances_obj_pair_values() for easy ways to find objects in this array and their corresponding values.

#### 23.8.2.4 hwloc_uint64_t* hwloc_distances_s::values

Matrix of distances between objects, stored as a one-dimension array. Distance from i-th to j-th object is stored in slot i*nbobjs+j. The meaning of the value depends on the `kind` attribute.

The documentation for this struct was generated from the following file:

- distances.h

## 23.9 hwloc_obj_attr_u::hwloc_group_attr_s Struct Reference

Group-specific Object Attributes.

`#include <hwloc.h>`

### Data Fields

- unsigned depth
- unsigned kind
- unsigned subkind
- unsigned char dont_merge

### 23.9.1 Detailed Description

Group-specific Object Attributes.

### 23.9.2 Field Documentation

#### 23.9.2.1 unsigned hwloc_obj_attr_u::hwloc_group_attr_s::depth

Depth of group object. It may change if intermediate Group objects are added.

#### 23.9.2.2 unsigned char hwloc_obj_attr_u::hwloc_group_attr_s::dont_merge

Flag preventing groups from being automatically merged with identical parent or children.

#### 23.9.2.3 unsigned hwloc_obj_attr_u::hwloc_group_attr_s::kind

Internally-used kind of group.

#### 23.9.2.4 unsigned hwloc_obj_attr_u::hwloc_group_attr_s::subkind

Internally-used subkind to distinguish different levels of groups with same kind.

The documentation for this struct was generated from the following file:

- hwloc.h

## 23.10 hwloc_info_s Struct Reference

Object info.

`#include <hwloc.h>`

### Data Fields

- char * name
- char * value

### 23.10.1 Detailed Description

Object info.

**See also:**

Consulting and Adding Key-Value Info Attributes

### 23.10.2 Field Documentation

#### 23.10.2.1 char* hwloc_info_s::name

Info name.

#### 23.10.2.2 char* hwloc_info_s::value

Info value.

The documentation for this struct was generated from the following file:

- hwloc.h

# 23.11 hwloc_obj_attr_u::hwloc_numanode_attr_s::hwloc_-memory_page_type_s Struct Reference

Array of local memory page types, `NULL` if no local memory and `page_types` is 0.

`#include <hwloc.h>`

## Data Fields

- hwloc_uint64_t size
- hwloc_uint64_t count

## 23.11.1 Detailed Description

Array of local memory page types, `NULL` if no local memory and `page_types` is 0. The array is sorted by increasing `size` fields. It contains `page_types_len` slots.

## 23.11.2 Field Documentation

### 23.11.2.1 hwloc_uint64_t hwloc_obj_attr_u::hwloc_numanode_attr_s::hwloc_memory_page_-type_s::count

Number of pages of this size.

### 23.11.2.2 hwloc_uint64_t hwloc_obj_attr_u::hwloc_numanode_attr_s::hwloc_memory_page_-type_s::size

Size of pages.

The documentation for this struct was generated from the following file:

- hwloc.h

## 23.12 hwloc_obj_attr_u::hwloc_numanode_attr_s Struct Reference

NUMA node-specific Object Attributes.

`#include <hwloc.h>`

### Data Structures

- struct hwloc_memory_page_type_s

  *Array of local memory page types, `NULL` if no local memory and `page_types` is 0.*

### Data Fields

- hwloc_uint64_t local_memory
- unsigned page_types_len
- struct hwloc_obj_attr_u::hwloc_numanode_attr_s::hwloc_memory_page_type_s * page_types

### 23.12.1 Detailed Description

NUMA node-specific Object Attributes.

### 23.12.2 Field Documentation

#### 23.12.2.1 hwloc_uint64_t hwloc_obj_attr_u::hwloc_numanode_attr_s::local_memory

Local memory (in bytes).

#### 23.12.2.2 struct hwloc_obj_attr_u::hwloc_numanode_attr_s::hwloc_memory_page_type_s * hwloc_obj_attr_u::hwloc_numanode_attr_s::page_types

Array of local memory page types, `NULL` if no local memory and `page_types` is 0. The array is sorted by increasing `size` fields. It contains `page_types_len` slots.

#### 23.12.2.3 unsigned hwloc_obj_attr_u::hwloc_numanode_attr_s::page_types_len

Size of array `page_types`.

The documentation for this struct was generated from the following file:

- hwloc.h

## 23.13 hwloc_obj Struct Reference

Structure of a topology object.

`#include <hwloc.h>`

### Data Fields

- hwloc_obj_type_t type
- char * subtype
- unsigned os_index
- char * name
- hwloc_uint64_t total_memory
- union hwloc_obj_attr_u * attr
- int depth
- unsigned logical_index
- struct hwloc_obj * next_cousin
- struct hwloc_obj * prev_cousin
- struct hwloc_obj * parent
- unsigned sibling_rank
- struct hwloc_obj * next_sibling
- struct hwloc_obj * prev_sibling
- int symmetric_subtree
- hwloc_cpuset_t cpuset
- hwloc_cpuset_t complete_cpuset
- hwloc_nodeset_t nodeset
- hwloc_nodeset_t complete_nodeset
- struct hwloc_info_s * infos
- unsigned infos_count
- void * userdata
- hwloc_uint64_t gp_index

**List and array of normal children below this object (except Memory, I/O and Misc children).**

- unsigned arity
- struct hwloc_obj ** children
- struct hwloc_obj * first_child
- struct hwloc_obj * last_child

**List of Memory children below this object.**

- unsigned memory_arity
- struct hwloc_obj * memory_first_child

**List of I/O children below this object.**

- unsigned io_arity
- struct hwloc_obj * io_first_child

**List of Misc children below this object.**

- unsigned misc_arity
- struct hwloc_obj * misc_first_child

### 23.13.1 Detailed Description

Structure of a topology object. Applications must not modify any field except `hwloc_obj.userdata`.

### 23.13.2 Field Documentation

#### 23.13.2.1 unsigned hwloc_obj::arity

Number of normal children. Memory, Misc and I/O children are not listed here but rather in their dedicated children list.

#### 23.13.2.2 union hwloc_obj_attr_u* hwloc_obj::attr [write]

Object type-specific Attributes, may be `NULL` if no attribute value was found.

#### 23.13.2.3 struct hwloc_obj** hwloc_obj::children [read]

Normal children, `children`[0 .. arity -1].

#### 23.13.2.4 hwloc_cpuset_t hwloc_obj::complete_cpuset

The complete CPU set of logical processors of this object,. This may include not only the same as the cpuset field, but also some CPUs for which topology information is unknown or incomplete, some offlines CPUs, and the CPUs that are ignored when the HWLOC_TOPOLOGY_FLAG_INCLUDE_DISALLOWED flag is not set. Thus no corresponding PU object may be found in the topology, because the precise position is undefined. It is however known that it would be somewhere under this object.

**Note:**

Its value must not be changed, hwloc_bitmap_dup() must be used instead.

#### 23.13.2.5 hwloc_nodeset_t hwloc_obj::complete_nodeset

The complete NUMA node set of this object,. This may include not only the same as the nodeset field, but also some NUMA nodes for which topology information is unknown or incomplete, some offlines nodes, and the nodes that are ignored when the HWLOC_TOPOLOGY_FLAG_INCLUDE_DISALLOWED flag is not set. Thus no corresponding NUMA node object may be found in the topology, because the precise position is undefined. It is however known that it would be somewhere under this object.

If there are no NUMA nodes in the machine, all the memory is close to this object, so only the first bit is set in `complete_nodeset`.

**Note:**

Its value must not be changed, hwloc_bitmap_dup() must be used instead.

#### 23.13.2.6 hwloc_cpuset_t hwloc_obj::cpuset

CPUs covered by this object. This is the set of CPUs for which there are PU objects in the topology under this object, i.e. which are known to be physically contained in this object and known how (the children path between this object and the PU objects).

If the HWLOC_TOPOLOGY_FLAG_INCLUDE_DISALLOWED configuration flag is set, some of these CPUs may not be allowed for binding, see hwloc_topology_get_allowed_cpuset().

**Note:**

All objects have non-NULL CPU and node sets except Misc and I/O objects.
Its value must not be changed, hwloc_bitmap_dup() must be used instead.

#### 23.13.2.7 int hwloc_obj::depth

Vertical index in the hierarchy. For normal objects, this is the depth of the horizontal level that contains this object and its cousins of the same type. If the topology is symmetric, this is equal to the parent depth plus one, and also equal to the number of parent/child links from the root object to here.

For special objects (NUMA nodes, I/O and Misc) that are not in the main tree, this is a special negative value that corresponds to their dedicated level, see hwloc_get_type_depth() and hwloc_get_type_depth_e. Those special values can be passed to hwloc functions such hwloc_get_nbobjs_by_depth() as usual.

#### 23.13.2.8 struct hwloc_obj* hwloc_obj::first_child [read]

First normal child.

#### 23.13.2.9 hwloc_uint64_t hwloc_obj::gp_index

Global persistent index. Generated by hwloc, unique across the topology (contrary to os_index) and persistent across topology changes (contrary to logical_index). Mostly used internally, but could also be used by application to identify objects.

#### 23.13.2.10 struct hwloc_info_s* hwloc_obj::infos [read]

Array of stringified info type=name.

#### 23.13.2.11 unsigned hwloc_obj::infos_count

Size of infos array.

#### 23.13.2.12 unsigned hwloc_obj::io_arity

Number of I/O children. These children are listed in `io_first_child`.

#### 23.13.2.13 struct hwloc_obj* hwloc_obj::io_first_child [read]

First I/O child. Bridges, PCI and OS devices are listed here (`io_arity` and `io_first_child`) instead of in the normal children list. See also hwloc_obj_type_is_io().

#### 23.13.2.14 struct hwloc_obj* hwloc_obj::last_child [read]

Last normal child.

#### 23.13.2.15 unsigned hwloc_obj::logical_index

Horizontal index in the whole list of similar objects, hence guaranteed unique across the entire machine. Could be a "cousin_rank" since it's the rank within the "cousin" list below Note that this index may change when restricting the topology or when inserting a group.

#### 23.13.2.16 unsigned hwloc_obj::memory_arity

Number of Memory children. These children are listed in `memory_first_child`.

#### 23.13.2.17 struct hwloc_obj* hwloc_obj::memory_first_child [read]

First Memory child. NUMA nodes and Memory-side caches are listed here (`memory_arity` and `memory_first_child`) instead of in the normal children list. See also hwloc_obj_type_is_memory(). A memory hierarchy starts from a normal CPU-side object (e.g. Package) and ends with NUMA nodes as leaves. There might exist some memory-side caches between them in the middle of the memory subtree.

#### 23.13.2.18 unsigned hwloc_obj::misc_arity

Number of Misc children. These children are listed in `misc_first_child`.

#### 23.13.2.19 struct hwloc_obj* hwloc_obj::misc_first_child [read]

First Misc child. Misc objects are listed here (`misc_arity` and `misc_first_child`) instead of in the normal children list.

#### 23.13.2.20 char* hwloc_obj::name

Object-specific name if any. Mostly used for identifying OS devices and Misc objects where a name string is more useful than numerical indexes.

#### 23.13.2.21 struct hwloc_obj* hwloc_obj::next_cousin [read]

Next object of same type and depth.

#### 23.13.2.22 struct hwloc_obj* hwloc_obj::next_sibling [read]

Next object below the same parent (inside the same list of children).

#### 23.13.2.23 hwloc_nodeset_t hwloc_obj::nodeset

NUMA nodes covered by this object or containing this object. This is the set of NUMA nodes for which there are NUMA node objects in the topology under or above this object, i.e. which are known to be

physically contained in this object or containing it and known how (the children path between this object and the NUMA node objects).

In the end, these nodes are those that are close to the current object.

If the HWLOC_TOPOLOGY_FLAG_INCLUDE_DISALLOWED configuration flag is set, some of these nodes may not be allowed for allocation, see hwloc_topology_get_allowed_nodeset().

If there are no NUMA nodes in the machine, all the memory is close to this object, so only the first bit may be set in `nodeset`.

**Note:**

All objects have non-NULL CPU and node sets except Misc and I/O objects.
Its value must not be changed, hwloc_bitmap_dup() must be used instead.

#### 23.13.2.24 unsigned hwloc_obj::os_index

OS-provided physical index number. It is not guaranteed unique across the entire machine, except for PUs and NUMA nodes. Set to HWLOC_UNKNOWN_INDEX if unknown or irrelevant for this object.

#### 23.13.2.25 struct hwloc_obj* hwloc_obj::parent [read]

Parent, `NULL` if root (Machine object).

#### 23.13.2.26 struct hwloc_obj* hwloc_obj::prev_cousin [read]

Previous object of same type and depth.

#### 23.13.2.27 struct hwloc_obj* hwloc_obj::prev_sibling [read]

Previous object below the same parent (inside the same list of children).

#### 23.13.2.28 unsigned hwloc_obj::sibling_rank

Index in parent's `children`[] array. Or the index in parent's Memory, I/O or Misc children list.

#### 23.13.2.29 char* hwloc_obj::subtype

Subtype string to better describe the type field.

#### 23.13.2.30 int hwloc_obj::symmetric_subtree

Set if the subtree of normal objects below this object is symmetric, which means all normal children and their children have identical subtrees. Memory, I/O and Misc children are ignored.

If set in the topology root object, lstopo may export the topology as a synthetic string.

#### 23.13.2.31 hwloc_uint64_t hwloc_obj::total_memory

Total memory (in bytes) in NUMA nodes below this object.

#### 23.13.2.32 hwloc_obj_type_t hwloc_obj::type

Type of object.

#### 23.13.2.33 void* hwloc_obj::userdata

Application-given private data pointer, initialized to `NULL`, use it as you wish. See hwloc_topology_set_-userdata_export_callback() in hwloc/export.h if you wish to export this field to XML.

The documentation for this struct was generated from the following file:

- hwloc.h

# 23.14 hwloc_obj_attr_u Union Reference

Object type-specific Attributes.

`#include <hwloc.h>`

## Data Structures

- struct hwloc_bridge_attr_s
  *Bridge specific Object Attribues.*
- struct hwloc_cache_attr_s
  *Cache-specific Object Attributes.*
- struct hwloc_group_attr_s
  *Group-specific Object Attributes.*
- struct hwloc_numanode_attr_s
  *NUMA node-specific Object Attributes.*
- struct hwloc_osdev_attr_s
  *OS Device specific Object Attributes.*
- struct hwloc_pcidev_attr_s
  *PCI Device specific Object Attributes.*

## Data Fields

- struct hwloc_obj_attr_u::hwloc_numanode_attr_s numanode
- struct hwloc_obj_attr_u::hwloc_cache_attr_s cache
- struct hwloc_obj_attr_u::hwloc_group_attr_s group
- struct hwloc_obj_attr_u::hwloc_pcidev_attr_s pcidev
- struct hwloc_obj_attr_u::hwloc_bridge_attr_s bridge
- struct hwloc_obj_attr_u::hwloc_osdev_attr_s osdev

### 23.14.1 Detailed Description

Object type-specific Attributes.

### 23.14.2 Field Documentation

#### 23.14.2.1 struct hwloc_obj_attr_u::hwloc_bridge_attr_s hwloc_obj_attr_u::bridge

Bridge specific Object Attribues.

#### 23.14.2.2 struct hwloc_obj_attr_u::hwloc_cache_attr_s hwloc_obj_attr_u::cache

Cache-specific Object Attributes.

#### 23.14.2.3 struct hwloc_obj_attr_u::hwloc_group_attr_s hwloc_obj_attr_u::group

Group-specific Object Attributes.

#### 23.14.2.4 struct hwloc_obj_attr_u::hwloc_numanode_attr_s hwloc_obj_attr_u::numanode

NUMA node-specific Object Attributes.

#### 23.14.2.5 struct hwloc_obj_attr_u::hwloc_osdev_attr_s hwloc_obj_attr_u::osdev

OS Device specific Object Attributes.

#### 23.14.2.6 struct hwloc_obj_attr_u::hwloc_pcidev_attr_s hwloc_obj_attr_u::pcidev

PCI Device specific Object Attributes.

The documentation for this union was generated from the following file:

- hwloc.h

# 23.15 hwloc_obj_attr_u::hwloc_osdev_attr_s Struct Reference

OS Device specific Object Attributes.

`#include <hwloc.h>`

## Data Fields

- hwloc_obj_osdev_type_t type

## 23.15.1 Detailed Description

OS Device specific Object Attributes.

## 23.15.2 Field Documentation

### 23.15.2.1 hwloc_obj_osdev_type_t hwloc_obj_attr_u::hwloc_osdev_attr_s::type

The documentation for this struct was generated from the following file:

- hwloc.h

## 23.16 hwloc_obj_attr_u::hwloc_pcidev_attr_s Struct Reference

PCI Device specific Object Attributes.

`#include <hwloc.h>`

### Data Fields

- unsigned short domain
- unsigned char bus
- unsigned char dev
- unsigned char func
- unsigned short class_id
- unsigned short vendor_id
- unsigned short device_id
- unsigned short subvendor_id
- unsigned short subdevice_id
- unsigned char revision
- float linkspeed

### 23.16.1 Detailed Description

PCI Device specific Object Attributes.

### 23.16.2 Field Documentation

#### 23.16.2.1 unsigned char hwloc_obj_attr_u::hwloc_pcidev_attr_s::bus

#### 23.16.2.2 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::class_id

#### 23.16.2.3 unsigned char hwloc_obj_attr_u::hwloc_pcidev_attr_s::dev

#### 23.16.2.4 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::device_id

#### 23.16.2.5 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::domain

#### 23.16.2.6 unsigned char hwloc_obj_attr_u::hwloc_pcidev_attr_s::func

#### 23.16.2.7 float hwloc_obj_attr_u::hwloc_pcidev_attr_s::linkspeed

#### 23.16.2.8 unsigned char hwloc_obj_attr_u::hwloc_pcidev_attr_s::revision

#### 23.16.2.9 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::subdevice_id

#### 23.16.2.10 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::subvendor_id

#### 23.16.2.11 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::vendor_id

The documentation for this struct was generated from the following file:

- hwloc.h

# 23.17 hwloc_topology_cpubind_support Struct Reference

Flags describing actual PU binding support for this topology.

`#include <hwloc.h>`

## Data Fields

- unsigned char set_thisproc_cpubind
- unsigned char get_thisproc_cpubind
- unsigned char set_proc_cpubind
- unsigned char get_proc_cpubind
- unsigned char set_thisthread_cpubind
- unsigned char get_thisthread_cpubind
- unsigned char set_thread_cpubind
- unsigned char get_thread_cpubind
- unsigned char get_thisproc_last_cpu_location
- unsigned char get_proc_last_cpu_location
- unsigned char get_thisthread_last_cpu_location

### 23.17.1 Detailed Description

Flags describing actual PU binding support for this topology. A flag may be set even if the feature isn't supported in all cases (e.g. binding to random sets of non-contiguous objects).

### 23.17.2 Field Documentation

#### 23.17.2.1 unsigned char hwloc_topology_cpubind_support::get_proc_cpubind

Getting the binding of a whole given process is supported.

#### 23.17.2.2 unsigned char hwloc_topology_cpubind_support::get_proc_last_cpu_location

Getting the last processors where a whole process ran is supported

#### 23.17.2.3 unsigned char hwloc_topology_cpubind_support::get_thisproc_cpubind

Getting the binding of the whole current process is supported.

#### 23.17.2.4 unsigned char hwloc_topology_cpubind_support::get_thisproc_last_cpu_location

Getting the last processors where the whole current process ran is supported

#### 23.17.2.5 unsigned char hwloc_topology_cpubind_support::get_thisthread_cpubind

Getting the binding of the current thread only is supported.

#### 23.17.2.6 unsigned char hwloc_topology_cpubind_support::get_thisthread_last_cpu_location

Getting the last processors where the current thread ran is supported

#### 23.17.2.7 unsigned char hwloc_topology_cpubind_support::get_thread_cpubind

Getting the binding of a given thread only is supported.

#### 23.17.2.8 unsigned char hwloc_topology_cpubind_support::set_proc_cpubind

Binding a whole given process is supported.

#### 23.17.2.9 unsigned char hwloc_topology_cpubind_support::set_thisproc_cpubind

Binding the whole current process is supported.

#### 23.17.2.10 unsigned char hwloc_topology_cpubind_support::set_thisthread_cpubind

Binding the current thread only is supported.

#### 23.17.2.11 unsigned char hwloc_topology_cpubind_support::set_thread_cpubind

Binding a given thread only is supported.

The documentation for this struct was generated from the following file:

- hwloc.h

# 23.18 hwloc_topology_diff_u::hwloc_topology_diff_generic_s Struct Reference

`#include <diff.h>`

## Data Fields

- hwloc_topology_diff_type_t type
- union hwloc_topology_diff_u * next

## 23.18.1 Field Documentation

### 23.18.1.1 union hwloc_topology_diff_u* hwloc_topology_diff_u::hwloc_topology_diff_generic_s::next [write]

### 23.18.1.2 hwloc_topology_diff_type_t hwloc_topology_diff_u::hwloc_topology_diff_generic_s::type

The documentation for this struct was generated from the following file:

- diff.h

## 23.19 hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_generic_s Struct Reference

`#include <diff.h>`

### Data Fields

- hwloc_topology_diff_obj_attr_type_t type

### 23.19.1 Field Documentation

#### 23.19.1.1 hwloc_topology_diff_obj_attr_type_t hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_generic_s::type

The documentation for this struct was generated from the following file:

- diff.h

# 23.20 hwloc_topology_diff_u::hwloc_topology_diff_obj_attr_s Struct Reference

`#include <diff.h>`

## Data Fields

- hwloc_topology_diff_type_t type
- union hwloc_topology_diff_u * next
- int obj_depth
- unsigned obj_index
- union hwloc_topology_diff_obj_attr_u diff

## 23.20.1 Field Documentation

### 23.20.1.1 union hwloc_topology_diff_obj_attr_u hwloc_topology_diff_u::hwloc_topology_diff_obj_attr_s::diff [write]

### 23.20.1.2 union hwloc_topology_diff_u* hwloc_topology_diff_u::hwloc_topology_diff_obj_attr_s::next [write]

### 23.20.1.3 int hwloc_topology_diff_u::hwloc_topology_diff_obj_attr_s::obj_depth

### 23.20.1.4 unsigned hwloc_topology_diff_u::hwloc_topology_diff_obj_attr_s::obj_index

### 23.20.1.5 hwloc_topology_diff_type_t hwloc_topology_diff_u::hwloc_topology_diff_obj_attr_s::type

The documentation for this struct was generated from the following file:

- diff.h

# 23.21 hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_string_s Struct Reference

String attribute modification with an optional name.

`#include <diff.h>`

## Data Fields

- hwloc_topology_diff_obj_attr_type_t type
- char * name
- char * oldvalue
- char * newvalue

### 23.21.1 Detailed Description

String attribute modification with an optional name.

### 23.21.2 Field Documentation

#### 23.21.2.1 char* hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_string_s::name

#### 23.21.2.2 char* hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_string_s::newvalue

#### 23.21.2.3 char* hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_string_s::oldvalue

#### 23.21.2.4 hwloc_topology_diff_obj_attr_type_t hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_string_s::type

The documentation for this struct was generated from the following file:

- diff.h

# 23.22 hwloc_topology_diff_obj_attr_u Union Reference

One object attribute difference.

`#include <diff.h>`

## Data Structures

- struct hwloc_topology_diff_obj_attr_generic_s
- struct hwloc_topology_diff_obj_attr_string_s

  *String attribute modification with an optional name.*

- struct hwloc_topology_diff_obj_attr_uint64_s

  *Integer attribute modification with an optional index.*

## Data Fields

- struct hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_generic_s generic
- struct hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_uint64_s uint64
- struct hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_string_s string

### 23.22.1 Detailed Description

One object attribute difference.

### 23.22.2 Field Documentation

#### 23.22.2.1 struct hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_generic_s hwloc_topology_diff_obj_attr_u::generic

#### 23.22.2.2 struct hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_string_s hwloc_topology_diff_obj_attr_u::string

String attribute modification with an optional name.

#### 23.22.2.3 struct hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_uint64_s hwloc_topology_diff_obj_attr_u::uint64

Integer attribute modification with an optional index.

The documentation for this union was generated from the following file:

- diff.h

# 23.23 hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_uint64_s Struct Reference

Integer attribute modification with an optional index.

`#include <diff.h>`

## Data Fields

- hwloc_topology_diff_obj_attr_type_t type
- hwloc_uint64_t index
- hwloc_uint64_t oldvalue
- hwloc_uint64_t newvalue

### 23.23.1 Detailed Description

Integer attribute modification with an optional index.

### 23.23.2 Field Documentation

#### 23.23.2.1 hwloc_uint64_t hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_uint64_s::index

#### 23.23.2.2 hwloc_uint64_t hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_uint64_s::newvalue

#### 23.23.2.3 hwloc_uint64_t hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_uint64_s::oldvalue

#### 23.23.2.4 hwloc_topology_diff_obj_attr_type_t hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_uint64_s::type

The documentation for this struct was generated from the following file:

- diff.h

# 23.24 hwloc_topology_diff_u::hwloc_topology_diff_too_complex_s Struct Reference

`#include <diff.h>`

## Data Fields

- hwloc_topology_diff_type_t type
- union hwloc_topology_diff_u * next
- int obj_depth
- unsigned obj_index

## 23.24.1 Field Documentation

### 23.24.1.1 union hwloc_topology_diff_u* hwloc_topology_diff_u::hwloc_topology_diff_too_complex_s::next [write]

### 23.24.1.2 int hwloc_topology_diff_u::hwloc_topology_diff_too_complex_s::obj_depth

### 23.24.1.3 unsigned hwloc_topology_diff_u::hwloc_topology_diff_too_complex_s::obj_index

### 23.24.1.4 hwloc_topology_diff_type_t hwloc_topology_diff_u::hwloc_topology_diff_too_complex_s::type

The documentation for this struct was generated from the following file:

- diff.h

## 23.25 hwloc_topology_diff_u Union Reference

One element of a difference list between two topologies.

`#include <diff.h>`

### Data Structures

- struct hwloc_topology_diff_generic_s
- struct hwloc_topology_diff_obj_attr_s
- struct hwloc_topology_diff_too_complex_s

### Data Fields

- struct hwloc_topology_diff_u::hwloc_topology_diff_generic_s generic
- struct hwloc_topology_diff_u::hwloc_topology_diff_obj_attr_s obj_attr
- struct hwloc_topology_diff_u::hwloc_topology_diff_too_complex_s too_complex

### 23.25.1 Detailed Description

One element of a difference list between two topologies.

### 23.25.2 Field Documentation

#### 23.25.2.1 struct hwloc_topology_diff_u::hwloc_topology_diff_generic_s hwloc_topology_diff_u::generic

#### 23.25.2.2 struct hwloc_topology_diff_u::hwloc_topology_diff_obj_attr_s hwloc_topology_diff_u::obj_attr

#### 23.25.2.3 struct hwloc_topology_diff_u::hwloc_topology_diff_too_complex_s hwloc_topology_diff_u::too_complex

The documentation for this union was generated from the following file:

- diff.h

# 23.26 hwloc_topology_discovery_support Struct Reference

Flags describing actual discovery support for this topology.

`#include <hwloc.h>`

## Data Fields

- unsigned char pu
- unsigned char numa
- unsigned char numa_memory
- unsigned char disallowed_pu
- unsigned char disallowed_numa

## 23.26.1 Detailed Description

Flags describing actual discovery support for this topology.

## 23.26.2 Field Documentation

### 23.26.2.1 unsigned char hwloc_topology_discovery_support::disallowed_numa

Detecting and identifying NUMA nodes that are not available to the current process is supported.

### 23.26.2.2 unsigned char hwloc_topology_discovery_support::disallowed_pu

Detecting and identifying PU objects that are not available to the current process is supported.

### 23.26.2.3 unsigned char hwloc_topology_discovery_support::numa

Detecting the number of NUMA nodes is supported.

### 23.26.2.4 unsigned char hwloc_topology_discovery_support::numa_memory

Detecting the amount of memory in NUMA nodes is supported.

### 23.26.2.5 unsigned char hwloc_topology_discovery_support::pu

Detecting the number of PU objects is supported.

The documentation for this struct was generated from the following file:

- hwloc.h

# 23.27 hwloc_topology_membind_support Struct Reference

Flags describing actual memory binding support for this topology.

`#include <hwloc.h>`

## Data Fields

- unsigned char set_thisproc_membind
- unsigned char get_thisproc_membind
- unsigned char set_proc_membind
- unsigned char get_proc_membind
- unsigned char set_thisthread_membind
- unsigned char get_thisthread_membind
- unsigned char set_area_membind
- unsigned char get_area_membind
- unsigned char alloc_membind
- unsigned char firsttouch_membind
- unsigned char bind_membind
- unsigned char interleave_membind
- unsigned char nexttouch_membind
- unsigned char migrate_membind
- unsigned char get_area_memlocation

## 23.27.1 Detailed Description

Flags describing actual memory binding support for this topology. A flag may be set even if the feature isn't supported in all cases (e.g. binding to random sets of non-contiguous objects).

## 23.27.2 Field Documentation

### 23.27.2.1 unsigned char hwloc_topology_membind_support::alloc_membind

Allocating a bound memory area is supported.

### 23.27.2.2 unsigned char hwloc_topology_membind_support::bind_membind

Bind policy is supported.

### 23.27.2.3 unsigned char hwloc_topology_membind_support::firsttouch_membind

First-touch policy is supported.

### 23.27.2.4 unsigned char hwloc_topology_membind_support::get_area_membind

Getting the binding of a given memory area is supported.

#### 23.27.2.5 unsigned char hwloc_topology_membind_support::get_area_memlocation

Getting the last NUMA nodes where a memory area was allocated is supported

#### 23.27.2.6 unsigned char hwloc_topology_membind_support::get_proc_membind

Getting the binding of a whole given process is supported.

#### 23.27.2.7 unsigned char hwloc_topology_membind_support::get_thisproc_membind

Getting the binding of the whole current process is supported.

#### 23.27.2.8 unsigned char hwloc_topology_membind_support::get_thisthread_membind

Getting the binding of the current thread only is supported.

#### 23.27.2.9 unsigned char hwloc_topology_membind_support::interleave_membind

Interleave policy is supported.

#### 23.27.2.10 unsigned char hwloc_topology_membind_support::migrate_membind

Migration flags is supported.

#### 23.27.2.11 unsigned char hwloc_topology_membind_support::nexttouch_membind

Next-touch migration policy is supported.

#### 23.27.2.12 unsigned char hwloc_topology_membind_support::set_area_membind

Binding a given memory area is supported.

#### 23.27.2.13 unsigned char hwloc_topology_membind_support::set_proc_membind

Binding a whole given process is supported.

#### 23.27.2.14 unsigned char hwloc_topology_membind_support::set_thisproc_membind

Binding the whole current process is supported.

#### 23.27.2.15 unsigned char hwloc_topology_membind_support::set_thisthread_membind

Binding the current thread only is supported.

The documentation for this struct was generated from the following file:

- hwloc.h

## 23.28 hwloc_topology_support Struct Reference

Set of flags describing actual support for this topology.

`#include <hwloc.h>`

### Data Fields

- struct hwloc_topology_discovery_support * discovery
- struct hwloc_topology_cpubind_support * cpubind
- struct hwloc_topology_membind_support * membind

### 23.28.1 Detailed Description

Set of flags describing actual support for this topology. This is retrieved with hwloc_topology_get_support() and will be valid until the topology object is destroyed. Note: the values are correct only after discovery.

### 23.28.2 Field Documentation

#### 23.28.2.1 struct hwloc_topology_cpubind_support* hwloc_topology_support::cpubind [read]

#### 23.28.2.2 struct hwloc_topology_discovery_support* hwloc_topology_support::discovery [read]

#### 23.28.2.3 struct hwloc_topology_membind_support* hwloc_topology_support::membind [read]

The documentation for this struct was generated from the following file:

- hwloc.h

# Index