Hardware Locality (hwloc)

1.11.3rc1

Generated by Doxygen 1.8.8

Wed Apr 13 2016 09:46:56

Contents

Chapter 1

Hardware Locality

Portable abstraction of hierarchical architectures for high-performance computing

1.1 Introduction

hwloc provides command line tools and a C API to obtain the hierarchical map of key computing elements, such as: NUMA memory nodes, shared caches, processor packages, processor 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. Additionally it may assemble the topologies of multiple machines into a single one so as to let applications consult the topology of an entire fabric or cluster at once.

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.

Note that the hwloc project represents the merger of the libtopology project from inria and the Portable Linux Processor Affinity (PLPA) sub-project from Open MPI. *Both of these prior projects are now deprecated.* The first hwloc release was essentially a "re-branding" of the libtopology code base, but with both a few genuinely new features and a few PLPA-like features added in. Prior releases of hwloc included documentation about switching from PLPA to hwloc; this documentation has been dropped on the assumption that everyone who was using PLPA has already switched to hwloc.

hwloc supports the following operating systems:

- Linux (including old kernels not having sysfs topology information, with knowledge of cpusets, offline CPUs, ScaleMP vSMP and Kerrighed support) on all supported hardware, including Intel Xeon Phi (KNL and KNC, either standalone or as a coprocessor) and NumaScale NumaConnect.
- Solaris
- AIX
- Darwin / OS X
- FreeBSD and its variants (such as kFreeBSD/GNU)
- NetBSD
- OSF/1 (a.k.a., Tru64)
- 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 only exception to this is kFreeBSD, which does not support topology information, and hwloc thus 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 below.

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
- Remote machine simulation through the gathering of Linux sysfs topology 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 CLI 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`).

1.2 Installation

hwloc (`http://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 (`http://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.

Nightly development snapshots are available on the web site. 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 >=1.10 and Libtool >=2.2.6 are required when building from a Git clone.

Installation by itself is the fairly common GNU-based process:

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

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:

- libnuma for memory binding and migration support on Linux (`numactl-devel` or `libnuma-dev` package).
- 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.
- the AMD OpenCL implementation for OpenCL device discovery.
- the NVIDIA CUDA Toolkit for CUDA device discovery.
- the NVIDIA Tesla Development Kit for NVML device discovery.
- the NV-CONTROL X extension library (NVCtrl) for NVIDIA display discovery.
- 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. 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).

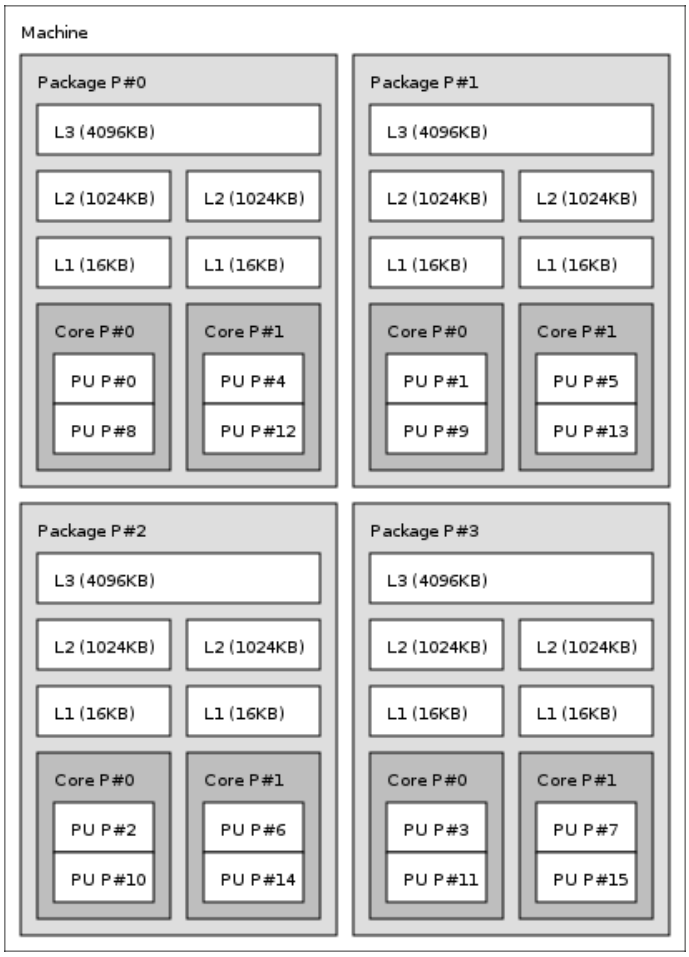
Also note that if you install supplemental libraries in non-standard locations, hwloc's configure script may not be able to find them without some help. You may need to specify additional CPPFLAGS, LDFLAGS, or PKG_CONFIG_PATH values on the configure command line.

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 ...
```

1.3 CLI Examples

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



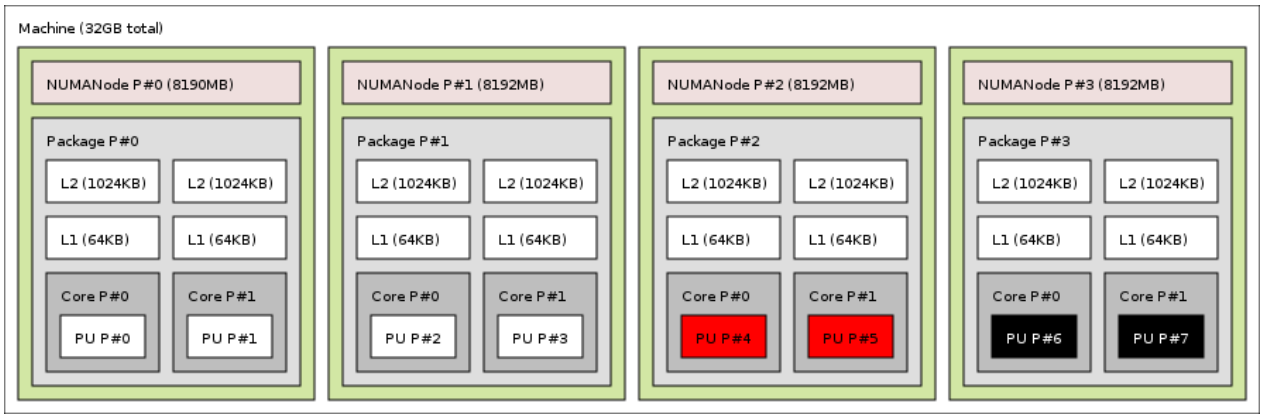
Here's the equivalent output in textual form:

```
Machine (16GB)
  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).

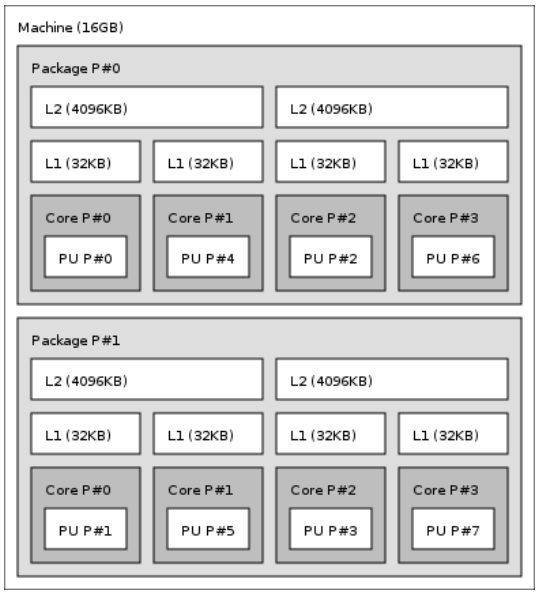
On a 4-package 2-core Opteron NUMA machine, the `lstopo` tool may show the following graphical output:



Here's the equivalent output in textual form:

```
Machine (32GB)
  NUMANode L#0 (P#0 8190MB) + Package L#0
    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)
  NUMANode L#1 (P#1 8192MB) + Package L#1
    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)
  NUMANode L#2 (P#2 8192MB) + Package L#2
    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)
  NUMANode L#3 (P#3 8192MB) + Package L#3
    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 (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)
```

1.4 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/.

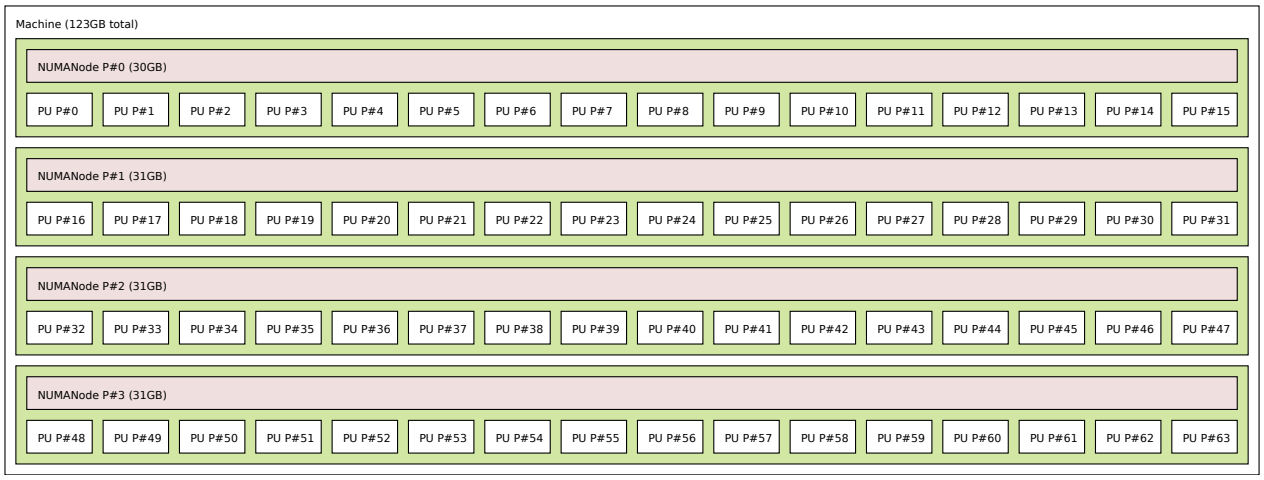
NOTE: If you are building the documentation from a Git clone, you will need to have Doxygen and pdflatex installed – the documentation will be built during the normal "make" process. The documentation is installed during "make install" to $prefix/share/doc/hwloc/ and your systems default man page tree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

1.4.1 Portability

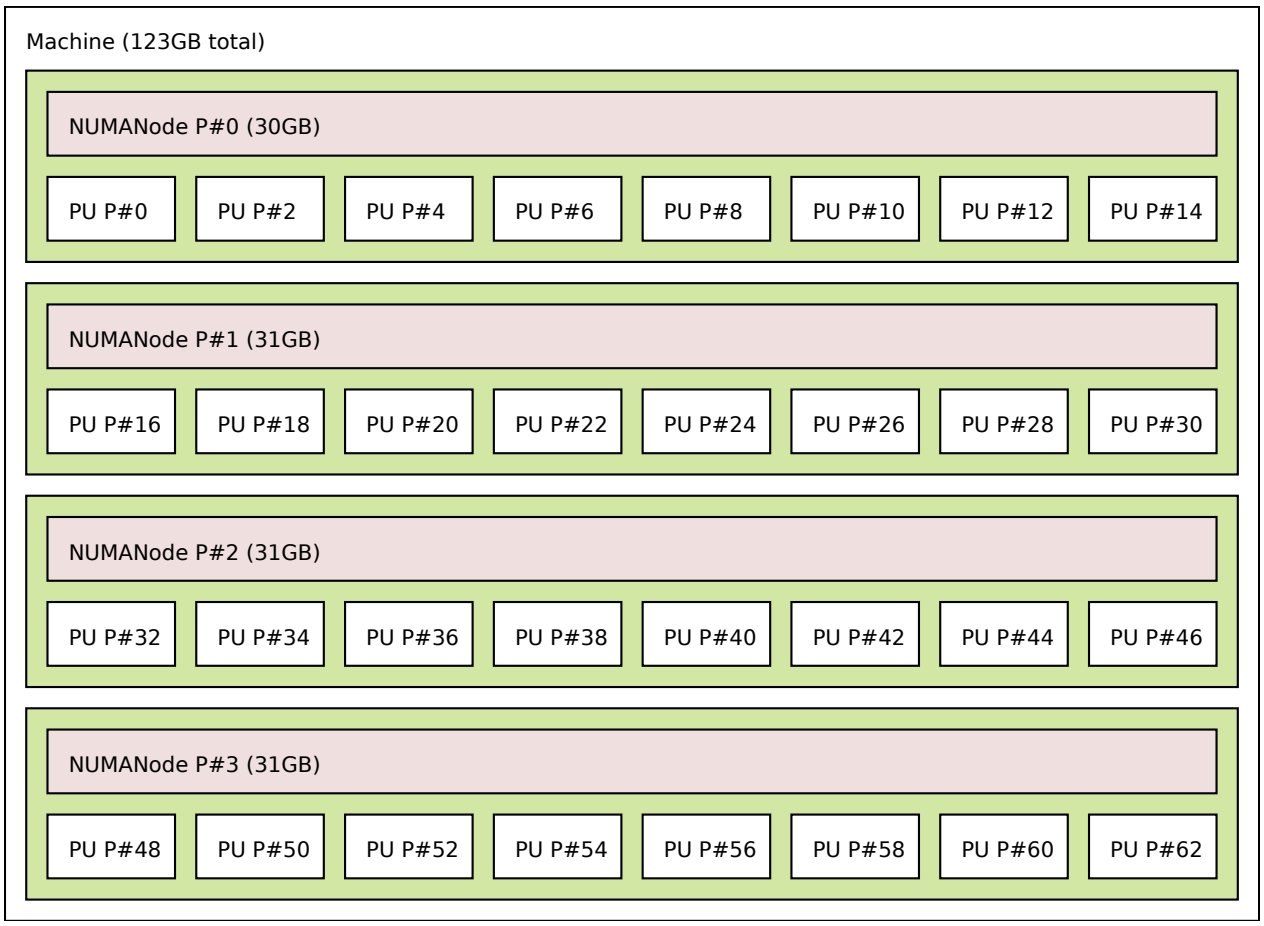
As shown in CLI Examples, hwloc can obtain information on a wide variety of hardware topologies. However, some platforms and/or operating system versions will only report a subset of this information. For example, on an PPC64-based system with 32 cores (each with 2 hardware threads) running a default 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.

Similarly, 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.

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:

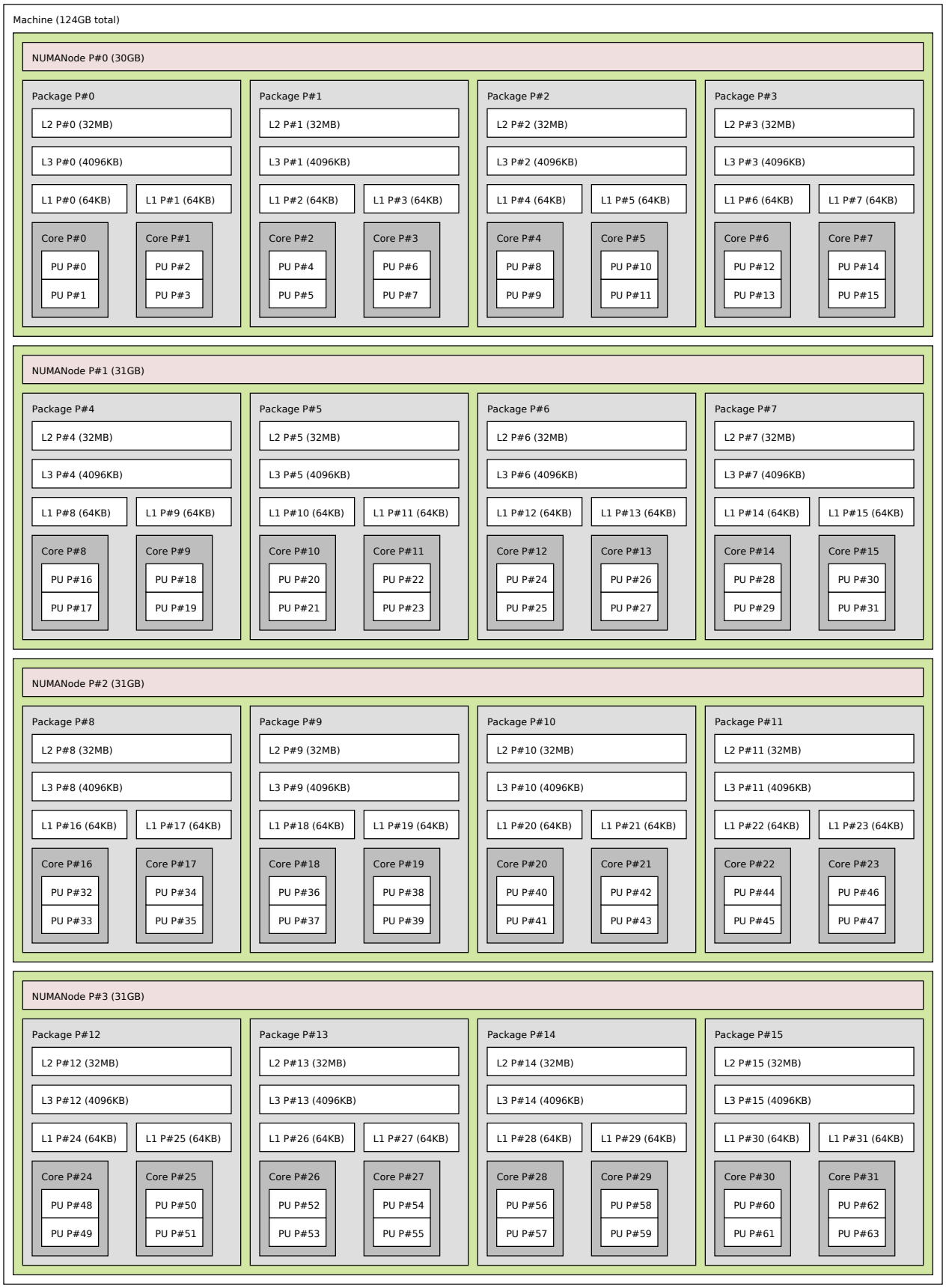


Notice that hwloc only sees half the PUs when SMT is disabled. PU #15, for example, seems to change location from NUMA node #0 to #1. In reality, no PUs "moved" – they were simply re-numbered when hwloc only saw half as many. Hence, PU #15 in the SMT-disabled picture probably corresponds to PU #30 in 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:


Machine (124GB total)
NUMANode P#0 (30GB)
Package P#0
L2 P#0 (32MB)
L3 P#0 (4096KB)
L1 P#0 (64KB)
L1 P#1 (64KB)
Core P#0
PU P#0
PU P#1
Core P#1
PU P#2
PU P#3
Package P#1
L2 P#1 (32MB)
L3 P#1 (4096KB)
L1 P#2 (64KB)
L1 P#3 (64KB)
Core P#2
PU P#4
PU P#5
Core P#3
PU P#6
PU P#7
Package P#2
L2 P#2 (32MB)
L3 P#2 (4096KB)
L1 P#4 (64KB)
L1 P#5 (64KB)
Core P#4
PU P#8
PU P#9
Core P#5
PU P#10
PU P#11
Package P#3
L2 P#3 (32MB)
L3 P#3 (4096KB)
L1 P#6 (64KB)
L1 P#7 (64KB)
Core P#6
PU P#12
PU P#13
Core P#7
PU P#14
PU P#15
NUMANode P#1 (31GB)
Package P#4
L2 P#4 (32MB)
L3 P#4 (4096KB)
L1 P#8 (64KB)
L1 P#9 (64KB)
Core P#8
PU P#16
PU P#17
Core P#9
PU P#18
PU P#19
Package P#5
L2 P#5 (32MB)
L3 P#5 (4096KB)
L1 P#10 (64KB)
L1 P#11 (64KB)
Core P#10
PU P#20
PU P#21
Core P#11
PU P#22
PU P#23
Package P#6
L2 P#6 (32MB)
L3 P#6 (4096KB)
L1 P#12 (64KB)
L1 P#13 (64KB)
Core P#12
PU P#24
PU P#25
Core P#13
PU P#26
PU P#27
Package P#7
L2 P#7 (32MB)
L3 P#7 (4096KB)
L1 P#14 (64KB)
L1 P#15 (64KB)
Core P#14
PU P#28
PU P#29
Core P#15
PU P#30
PU P#31
NUMANode P#2 (31GB)
Package P#8
L2 P#8 (32MB)
L3 P#8 (4096KB)
L1 P#16 (64KB)
L1 P#17 (64KB)
Core P#16
PU P#32
PU P#33
Core P#17
PU P#34
PU P#35
Package P#9
L2 P#9 (32MB)
L3 P#9 (4096KB)
L1 P#18 (64KB)
L1 P#19 (64KB)
Core P#18
PU P#36
PU P#37
Core P#19
PU P#38
PU P#39
Package P#10
L2 P#10 (32MB)
L3 P#10 (4096KB)
L1 P#20 (64KB)
L1 P#21 (64KB)
Core P#20
PU P#40
PU P#41
Core P#21
PU P#42
PU P#43
Package P#11
L2 P#11 (32MB)
L3 P#11 (4096KB)
L1 P#22 (64KB)
L1 P#23 (64KB)
Core P#22
PU P#44
PU P#45
Core P#23
PU P#46
PU P#47
NUMANode P#3 (31GB)
Package P#12
L2 P#12 (32MB)
L3 P#12 (4096KB)
L1 P#24 (64KB)
L1 P#25 (64KB)
Core P#24
PU P#48
PU P#49
Core P#25
PU P#50
PU P#51
Package P#13
L2 P#13 (32MB)
L3 P#13 (4096KB)
L1 P#26 (64KB)
L1 P#27 (64KB)
Core P#26
PU P#52
PU P#53
Core P#27
PU P#54
PU P#55
Package P#14
L2 P#14 (32MB)
L3 P#14 (4096KB)
L1 P#28 (64KB)
L1 P#29 (64KB)
Core P#28
PU P#56
PU P#57
Core P#29
PU P#58
PU P#59
Package P#15
L2 P#15 (32MB)
L3 P#15 (4096KB)
L1 P#30 (64KB)
L1 P#31 (64KB)
Core P#30
PU P#60
PU P#61
Core P#31
PU P#62
PU P#63

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, there may someday be an element "lower" than a PU, or perhaps a new element may exist between a core and a PU.

1.4.2 API Example

The following small C example (named “hwloc-hello.c”) prints the topology of the machine and bring the process to the first logical processor of the second core of the machine. More examples are available in the doc/examples/ directory of the source tree.

```
/* Example hwloc API program.
 *
 * See other examples under doc/examples/ in the source tree
 * for more details.
 *
 * Copyright © 2009-2015 Inria.  All rights reserved.
 * Copyright © 2009-2011 Université Bordeaux
 * Copyright © 2009-2010 Cisco Systems, Inc.  All rights reserved.
 * See COPYING in top-level directory.
 *
 * hwloc-hello.c
 */

#include <hwloc.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>

static void print_children(hwloc_topology_t topology,
      hwloc_obj_t obj,
                           int depth)
{
    char type[32], attr[1024];
    unsigned i;

    hwloc_obj_type_snprintf(type, sizeof(type), obj, 0);
    printf("%*s%s", 2*depth, "", type);
    if (obj->os_index != (unsigned) -1)
      printf("#%u", obj->os_index);
    hwloc_obj_attr_snprintf(attr, sizeof(attr), obj, " ", 0);
    if (*attr)
      printf("(%s)", attr);
    printf("\n");
    for (i = 0; i < obj->arity; i++) {
        print_children(topology, obj->children[i], depth + 1);
    }
}

int main(void)
{
    int depth;
    unsigned i, n;
    unsigned long size;
    int levels;
    char string[128];
    int topodepth;
    hwloc_topology_t topology;
    hwloc_cpuset_t cpuset;
    hwloc_obj_t obj;

    /* Allocate and initialize topology object. */
    hwloc_topology_init(&topology);

    /* ... Optionally, put detection configuration here to ignore
       some objects types, define a synthetic topology, etc....
```

```
   The default is to detect all the objects of the machine that
   the caller is allowed to access.  See Configure Topology
   Detection. */

/* Perform the topology detection. */
hwloc_topology_load(topology);

/* Optionally, get some additional topology information
   in case we need the topology depth later. */
topodepth = hwloc_topology_get_depth(topology);

/*****************************************************************
 * First example:
 * Walk the topology with an array style, from level 0 (always
 * the system level) to the lowest level (always the proc level).
 *****************************************************************/
for (depth = 0; depth < topodepth; depth++) {
    printf("*** Objects at level %d\n", depth);
    for (i = 0; i < hwloc_get_nbobjs_by_depth(topology, depth);
         i++) {
        hwloc_obj_type_snprintf(string, sizeof(string),
                                hwloc_get_obj_by_depth(topology, depth, i), 0);
        printf("Index %u: %s\n", i, string);
    }
}

/*****************************************************************
 * Second example:
 * Walk the topology with a tree style.
 *****************************************************************/
printf("*** Printing overall tree\n");
print_children(topology, hwloc_get_root_obj(topology), 0);

/*****************************************************************
 * Third example:
 * Print the number of packages.
 *****************************************************************/
depth = hwloc_get_type_depth(topology, HWLOC_OBJ_PACKAGE);
if (depth == HWLOC_TYPE_DEPTH_UNKNOWN) {
    printf("*** The number of packages is unknown\n");
} else {
    printf("*** %u package(s)\n",
           hwloc_get_nbobjs_by_depth(topology, depth));
}

/*****************************************************************
 * Fourth example:
 * Compute the amount of cache that the first logical processor
 * has above it.
 *****************************************************************/
levels = 0;
size = 0;
for (obj = hwloc_get_obj_by_type(topology, HWLOC_OBJ_PU, 0);
     obj;
     obj = obj->parent)
  if (obj->type == HWLOC_OBJ_CACHE) {
    levels++;
    size += obj->attr->cache.size;
  }
printf("*** Logical processor 0 has %d caches totaling %luKB\n",
       levels, size / 1024);

/*****************************************************************
 * Fifth example:
 * Bind to only one thread of the last core of the machine.
 *
 * First find out where cores are, or else smaller sets of CPUs if
 * the OS doesn't have the notion of a "core".
 *****************************************************************/
depth = hwloc_get_type_or_below_depth(topology,
  HWLOC_OBJ_CORE);

/* Get last core. */
obj = hwloc_get_obj_by_depth(topology, depth,
                hwloc_get_nbobjs_by_depth(topology, depth) - 1);
if (obj) {
    /* Get a copy of its cpuset that we may modify. */
    cpuset = hwloc_bitmap_dup(obj->cpuset);

    /* Get only one logical processor (in case the core is
       SMT/hyper-threaded). */
```

```
        hwloc_bitmap_singlify(cpuset);

        /* And try to bind ourself there. */
        if (hwloc_set_cpubind(topology, cpuset, 0)) {
            char *str;
            int error = errno;
            hwloc_bitmap_asprintf(&str, obj->cpuset);
            printf("Couldn't bind to cpuset %s: %s\n", str, strerror(error));
            free(str);
        }

        /* Free our cpuset copy */
        hwloc_bitmap_free(cpuset);
    }

    /************************************************************
     * Sixth example:
     * Allocate some memory on the last NUMA node, bind some existing
     * memory to the last NUMA node.
     ************************************************************/
    /* Get last node. */
    n = hwloc_get_nbobjs_by_type(topology,
      HWLOC_OBJ_NUMANODE);
    if (n) {
        void *m;
        size = 1024*1024;

        obj = hwloc_get_obj_by_type(topology,
      HWLOC_OBJ_NUMANODE, n - 1);
        m = hwloc_alloc_membind_nodeset(topology, size, obj->
      nodeset,
                HWLOC_MEMBIND_BIND, 0);
        hwloc_free(topology, m, size);

        m = malloc(size);
        hwloc_set_area_membind_nodeset(topology, m, size, obj->
      nodeset,
                HWLOC_MEMBIND_BIND, 0);
        free(m);
    }

    /* Destroy topology object. */
    hwloc_topology_destroy(topology);

    return 0;
}
```

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 += $(pkg-config --cflags hwloc)
LDLIBS += $(pkg-config --libs hwloc)
cc hwloc-hello.c $(CFLAGS) -o hwloc-hello $(LDLIBS)
```

On a machine with 4GB of RAM and 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(3938MB)
*** 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(3938MB)
  Package#0
    Core#0
      PU#0
    Core#1
      PU#1
  Package#1
    Core#3
      PU#2
    Core#2
      PU#3
*** 2 package(s)
shell$
```

1.5 Questions and Bugs

Questions should be sent to the devel mailing list (http://www.open-mpi.org/community/lists/hwloc.php). Bug reports should be reported in the tracker (https://github.com/open-mpi/hwloc/issues).

If hwloc discovers an incorrect topology for your machine, the very first thing you should check is to ensure that you have the most recent updates installed for your operating system. Indeed, most of hwloc topology discovery relies on hardware information retrieved through the operation system (e.g., via the /sys virtual filesystem of the Linux kernel). If upgrading your OS or Linux kernel does not solve your problem, you may also want to ensure that you are running the most recent version of the BIOS for your machine.

If those things fail, contact us on the mailing list for additional help. Please attach the output of lstopo after having given the `--enable-debug` option to ./configure and rebuilt completely, to get debugging output. Also attach the `/proc` + `/sys` tarball generated by the installed script `hwloc-gather-topology` when submitting problems about Linux, or send the output of `kstat cpu_info` in the Solaris case, or the output of `sysctl hw` in the Darwin or BSD cases.

1.6 History / Credits

hwloc is the evolution and merger of the libtopology (http://runtime.bordeaux.inria.fr/libtopology/) project and the Portable Linux Processor Affinity (PLPA) (http://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 (http://runtime.bordeaux.inria.fr/) (headed by Raymond Namyst (http://dept-info.labri.fr/~namyst/). 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.

1.7 Further Reading

The documentation chapters include

- Terms and Definitions
- Command-Line Tools
- Environment Variables

- CPU and Memory Binding Overview
- I/O Devices
- Miscellaneous objects
- Multi-node Topologies
- 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

Make sure to have had a look at those too!

Chapter 2

Terms and Definitions

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

They are topologically sorted by CPU set into a tree.

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.

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.

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.

Parent object The object logically containing the current object, for example because its CPU set includes the CPU set of the current object.

Ancestor object The parent object, or its own parent object, 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.

Arity The number of children of an object.

Sibling objects Objects which have the same parent. 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, parent_arity).

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.

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.

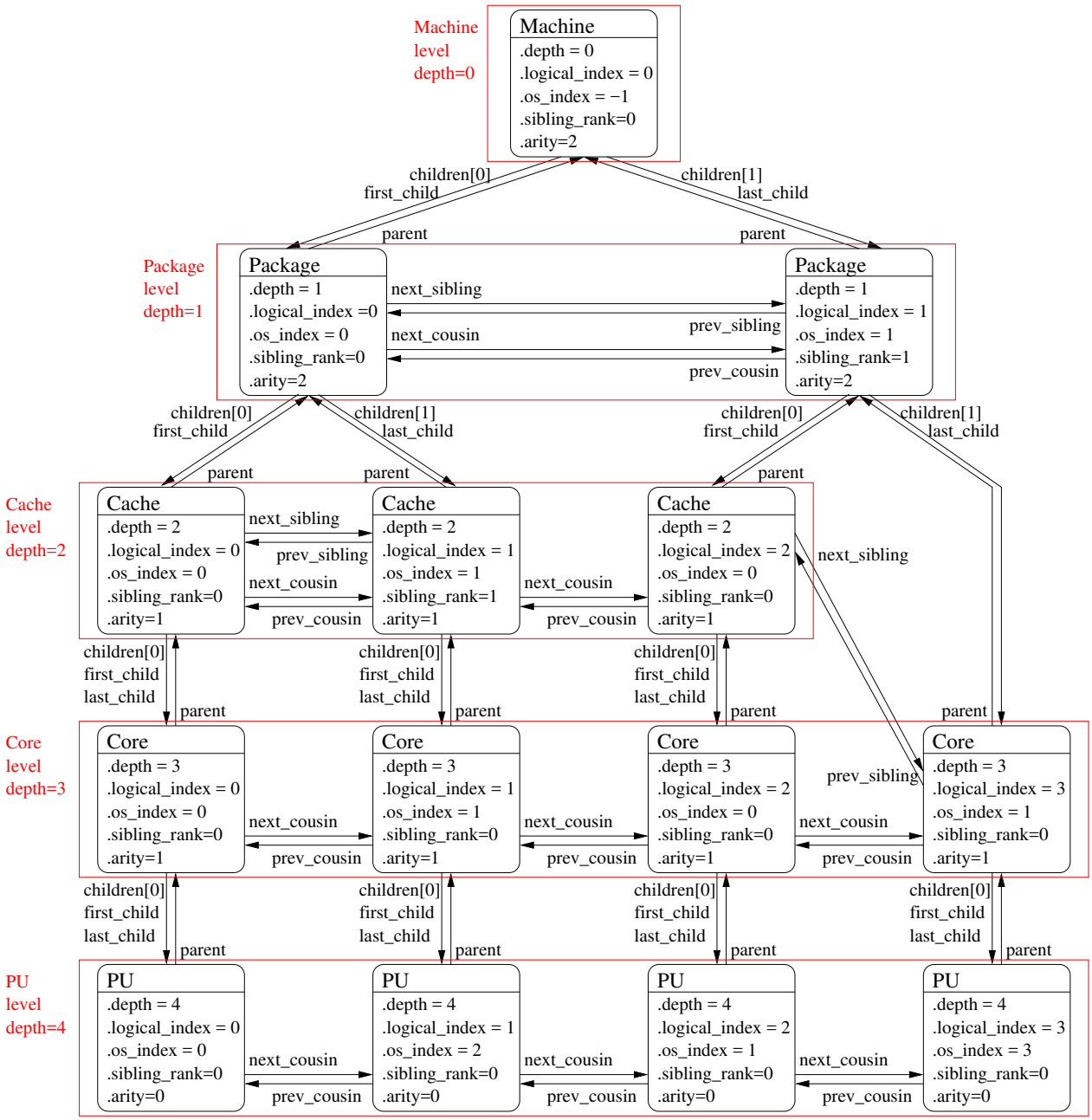
OS or physical index The index that the operating system (OS) uses to identify the object. This may be completely arbitrary, non-unique, non-contiguous, not representative of logical proximity, and may depend on the BIOS configuration. That is why hwloc almost never uses them, only in the default lstopo 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?.

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

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 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.

Chapter 3

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.

3.1 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).

3.2 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).

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

3.4 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.

3.5 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.

3.6 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.

3.7 hwloc-distances

hwloc-distances displays all distance matrices attached to the topology. Note that lstopo may also display distance matrices in its verbose textual output. However lstopo only prints matrices that cover the entire topology while hwloc-distances also displays matrices that ignore part of the topology.

3.8 hwloc-annotate

hwloc-annotate may add object 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.

3.9 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.

3.10 hwloc-assembler

hwloc-assembler combines several XML topology files into a single multi-node XML topology. It may then be used later as input with hwloc_topology_set_xml() or with the HWLOC_XMLFILE environment variable. See Multi-node Topologies for details.

3.11 hwloc-assembler-remote

hwloc-assembler-remote is a frontend to hwloc-assembler. It takes care of contacting the given list of remote hosts (through ssh) and retrieving their topologies as XML before assembling them with hwloc-assembler.

3.12 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 Knights Landing Xeon Phi platforms. See Why do I need hwloc-dump-hwdata for caches on Intel Knights Landing Xeon Phi?.

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

3.13 hwloc-gather-topology

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 output).

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

Chapter 4

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_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_FSROOT=/path/to/linux/filesystem-root/ switches to reading the topology from the specified Linux filesystem root instead of the main file-system root, as if hwloc_topology_set_fsroot() had been called. 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_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.

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_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, 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_<type>_DISTANCES=index,...:X*Y

HWLOC_<type>_DISTANCES=begin-end:X*Y*Z

HWLOC_<type>_DISTANCES=index,...:distance,... sets a distance matrix for objects of the given type and physical indexes. The type should be given as its case-sensitive stringified value (e.g. `NUMANode`, `Package`, `Cache`, `Core`, `PU`). If another distance matrix already exists for the given type, either because the user specified it or because the OS offers it, it will be replaced by the given one.

If the variable value is `none`, the existing distance matrix for the given type is removed. Otherwise, the variable value first consists in a list of physical indexes that may be specified as a comma-separated list (e.g. `0,2,4,1,3,5`) or as a range of consecutive indexes (`0-5`). It is followed by a colon and the corresponding distances:

- If `X*Y` is given, X groups of Y close objects are specified.
- If `X*Y*Z` is given, X groups of Y groups of Z close objects are specified.
- Otherwise, the comma-separated list of distances should be given. If N objects are considered, the i*N+j-th value gives the distance from the i-th object to the j-th object. These distance values must use a dot as a decimal separator.

Note that distances are ignored in multi-node topologies.

HWLOC_PCI_<domain>_<bus>_LOCALCPUS=<cpuset> changes the locality of I/O devices behind the specified PCI hostbridge. 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 (the entire set of objects behind a host bridge) near a custom set of processors. `domain` and `bus` are the PCI domain and primary bus of the corresponding host bridge.

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_V↩ ERBOSE=1.

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. 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 the variable

is set to an empty string, no specific component is loaded first, all components are loaded in priority order, this is strictly identical to not specifying any variable. The `xml` component name may be followed by a XML file to load (`xml=file.xml`). The synthetic component may be followed by a basic synthetic topology description (`synthetic=node:2 pu:3`, see Synthetic topologies). This variable does not take precedence over the application selecting components with functions such as hwloc_topology_set_xml(). See Components and plugins 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_DEBUG_VERBOSE=0 disables all verbose messages that are enabled by default when `-enable-debug` is passed to configure.

Chapter 5

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()
- functions that allocate memory bound to specific node set without changing the current memory binding policy (if supported): hwloc_alloc_membind() and hwloc_alloc_membind_nodeset().
- helpers which, if needed, change the current memory binding policy of the process in order to obtain memory binding: hwloc_alloc_membind_policy() and hwloc_alloc_membind_policy_nodeset()

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 6

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.

6.1 Enabling and requirements

I/O discovery is disabled by default (except in lstopo) so as not to break legacy application by adding unexpected I/O objects to the topology. It can be enabled by passing flags such as `HWLOC_TOPOLOGY_FLAG_IO_DEVICES` to hwloc_topology_set_flags() before loading the topology.

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. 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 reported I/O devices as being attached to the topology root object.

6.2 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 devices and bridges. See PCI devices and bridges.
- `HWLOC_OBJ_MISC` describes miscellaneous devices such as memory modules (DIMMs). These are neither PCI nor OS handles, therefore they appear as Misc objects. See Misc objects added by hwloc.

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 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.

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).

6.3 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) below their PCI 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 (HWLOC_OBJ_OSDEV_BLOCK)
  - *sda* (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)
  - *mic0* for the first Intel Xeon Phi (MIC) coprocessor (Linux component)
- DMA engine channel (HWLOC_OBJ_OSDEV_DMA)
  - *dma0chan0* (Linux component) when full I/O discovery is enabled (HWLOC_TOPOLOGY_FLAG_WHOLE_IO)

When none of the above strategies is supported and enabled, hwloc cannot place any OS object inside PCI objects. 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.

6.4 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.

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

6.5 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 regular 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 a NUMA 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.

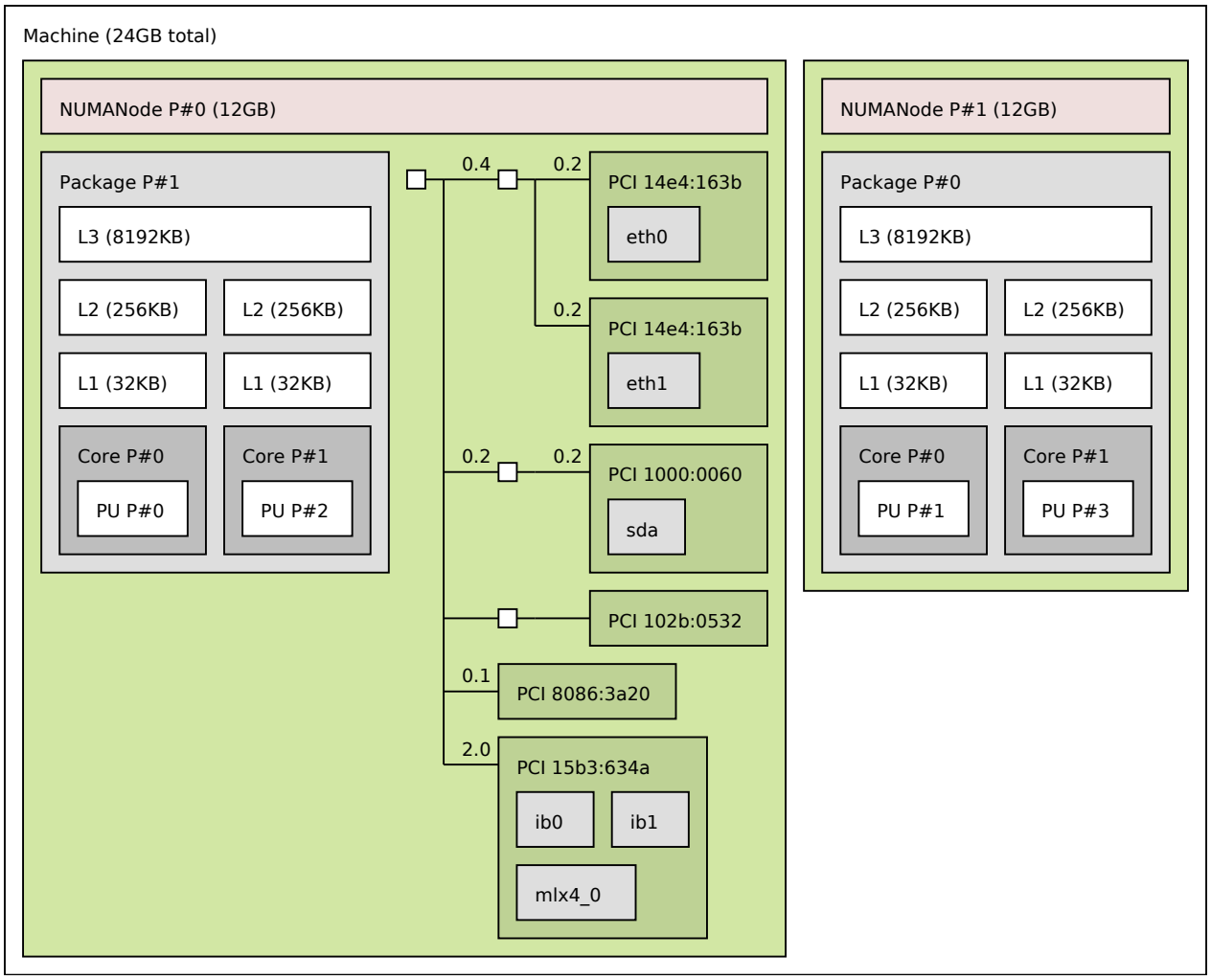
- `pci=0000:02:03.0` is replaced by the set of CPUs that are close to the PCI device whose bus ID is given.

- `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.

6.6 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*).

PCI link speed is also reported for some bridges and devices because lstopo was privileged when it discovered the topology.

Here is the corresponding textual output:

```
Machine (24GB)
  NUMANode L#0 (P#0 12GB)
    Package L#0 + 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 14e4:163b
          Net "eth0"
        PCI 14e4:163b
          Net "eth1"
```

```
      PCIBridge
        PCI 1000:0060
          Block "sda"
      PCIBridge
        PCI 102b:0532
      PCI 8086:3a20
      PCI 15b3:634a
        Net "ib0"
        Net "ib1"
        Net "mlx4_0"
  NUMANode L#1 (P#1 12GB) + Package L#1 + 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 7

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 (ee also How do I annotate the topology with private notes?).

These Misc objects may have a Type info attribute to replace Misc with something else in the lstopo output.

7.1 Misc objects added by hwloc

hwloc only uses Misc objects when other object types are not sufficient. This currently includes:

- Memory modules (DIMMs), on Linux when privileged and when dmi-sysfs is supported by the kernel, and when I/O discovery is enabled. These objects have a Type info attribute 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 Type=MemoryModule DeviceLocation="Bottom-Slot 2(right)" BankLocation="BANK 2"
      Vendor=Elpida SerialNumber=21733667 AssetTag=9876543210 PartNumber="EBJ81UG8EFU0-GN-F ")
```

- Displaying process binding in lstopo --top. These objects have a Type info attribute 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
```

7.2 Annotating topologies with Misc objects

The user may annotate hwloc topologies with its own Misc objects. A Misc object may be inserted anywhere in the topology by specifying its CPU set (using hwloc_topology_insert_misc_object_by_cpuset()). Or it may be inserted as a leaf of the topology by specifying its parent (with hwloc_topology_insert_misc_object_by↩_parent()).

Chapter 8

Multi-node Topologies

hwloc is usually used for consulting and manipulating single machine topologies. This includes large systems as long as a single instance of the operating system manages the entire system. However it is sometimes desirable to have multiple independent hosts inside the same topology, for instance when applying algorithms to an entire cluster topology. hwloc therefore offers the ability to agregate multiple host topologies into a single global one.

8.1 Multi-node Objects Specifities

A multi-node topology contains several single-node topologies. Those are assembled by making their own root objects (usually Machine object) children of higher objects. These higher objects include at least the root of the global topology (usually a System object). Some intermediate objects may also exists, for instance to represent switches in a large fabric.

There are actually three possible types of objects that have different properties with respect to cpusets, nodesets and binding. Indeed those cpusets and nodesets were designed for execution and memory binding within a single operating system. Binding on another system or across several different systems would be meaningless.

Local objects Any object that corresponds to the local machine may be manipulated as usual. Obviously, if the multi-node topology does not contain the local machine topology, no such local object exists.

Objects from other nodes Any object that comes from inside another node is represented as usual but its cpusets and nodesets should not be used for binding since binding on another system makes no sense.

Objects above single nodes Any object above single-node topologies does not have any cpuset or nodeset pointer because binding across multiple systems makes no sense. This includes the glocal root object of a multi-node topology and possibly some intermediate objects between this global root and the local root of single-node topologies.

It is important to keep this in mind before binding using multi-node topologies. To make sure binding on an object is possible, one should first check that its cpuset or nodeset pointer is not NULL. Then, one should check whether the object is indeed local.

To find out which machine a given object corresponds to, one may look at the info attributes of the parent Machine object. The `HostName` info is usually available in Machine objects, it may be retrieved with the following code:

```
hwloc_obj_t machine_obj;
obj = hwloc_get_ancestor_obj_by_type(topology, HWLOC_OBJ_MACHINE, obj);
if (machine_obj)
  return hwloc_obj_get_info_by_name(machine_obj, "HostName");
else
  return NULL;
```

The hwloc assembler scripts (see below) also add `AssemblerName` and `AssemblerIndex` info attributes to the Machine objects to identify the corresponding host name and index during assembly.

8.2 Assembling topologies with command-line tools

One way to manipulate multinode topologies is to retrieve other nodes' topologies as XML files and combine them as a global XML topology. It may then be loaded with hwloc_topology_set_xml() or with the HWLOC_XMLFILE environment variable.

The hwloc-assembler and hwloc-assembler-remote utilities offer the ability to combine XML topologies or remote nodes' topologies (see Command-Line Tools).

8.3 Assembling topologies with the programming interface

The hwloc programming interface offers the ability to build multinode topologies using the *custom* interface. A new multinode topology has to be initialized with hwloc_topology_init() and then set to custom with hwloc_topology_set_custom(). Topologies and objects mat then be assembled. Later, the custom topology is finalized as usual with hwloc_topology_load().

A custom topology starts with a single root object of type System. It may be modified by inserting a new child object with hwloc_custom_insert_group_object_by_parent() or by duplicating another topology with hwloc_custom_insert_topology(). Both of these operations require to specify the parent object in the custom topology where the insertion will take place. This parent may be either the root (returned by hwloc_get_root_obj()) or an already-inserted object (returned by hwloc_custom_insert_group_object_by_parent()).

Ideally, any existing object in the custom topology could be the parent. However, special care should be taken when traversing the topology to find such an object because most links between objects (children, siblings, cousins) are not setup until hwloc_topology_load() is invoked.

8.4 Example of assembly with the programming interface

If the topologies of two hosts have been previously gathered in XML files `host1.xml` and `host2.xml`, the global topology may be assembled with the following code.

```
hwloc_topology_t host1, host2, global;

/* initialize global topology */
hwloc_topology_init(&global);
hwloc_topology_set_custom(global);

/* insert host1 entire topology below the global topology root */
hwloc_topology_init(&host1);
hwloc_topology_load(host1);
hwloc_custom_insert_topology(global, hwloc_get_root_obj(global),
                             host1, NULL);
hwloc_topology_destroy(host1);

/* insert host2 entire topology below the global topology root */
hwloc_topology_init(&host2);
hwloc_topology_load(host2);
hwloc_custom_insert_topology(global, hwloc_get_root_obj(global),
                             host2, NULL);
hwloc_topology_destroy(host2);

/* load and play with the global topology */
```

```
hwloc_topology_load(global);
...
```

If a intermediate object such as a switch should be inserted above one of the host topologies:

```
...
/* insert a switch object below the global topology root */
hwloc_obj_t sw =
   hwloc_custom_insert_group_object_by_parent(global,
                              hwloc_get_root_obj(global), 0);

/* insert host2 entire topology below the switch */
hwloc_topology_init(&host2);
hwloc_topology_load(host2);
hwloc_custom_insert_topology(global, switch, host2, NULL);
hwloc_topology_destroy(host2);

/* load and play with the global topology */
hwloc_topology_load(global);
...
```

Chapter 9

Object attributes

9.1 Normal attributes

hwloc objects have many attributes. The hwloc_obj structure contains a common set of attributes that are available for object types, for instance their `type` or `logical_index`.

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 Cache object `obj` contains cache-specific information in `obj->attr->cache`, such as its size and associativity. See hwloc_obj_attr_u for details.

9.2 Custom string infos

Aside from the `name` field of each object, 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 (with the usual corresponding object in parentheses). 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.

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

Backend (Machine object or topology root object) 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`, `Linux` and `pci`.

LinuxCgroup (Machine object) The name the Linux control group where the calling process is placed.

SyntheticDescription (topology root object) The description string that was given to hwloc to build this synthetic topology.

CPUModel (Package or Machine) The processor model name. Usually added to Package objects, but can be in Machine instead if hwloc failed to discover any package.

CPUType (Package) A Solaris-specific general processor type name, such as "i86pc".

CPUVendor, CPUModelNumber, CPUFamilyNumber, CPUStepping (Package or Machine) 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). Usually added to Package objects, but can be in Machine instead if hwloc failed to discover any package.

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

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.

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

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

PCISlot The name/number of the physical slot where the PCI device is plugged.

Vendor, Model, Revision, SerialNumber The vendor and model names, revision, and serial number of a Block OS device.

LinuxDeviceID The major/minor device number such as 8:0 on Linux for a Block OS device.

CoProcType (Co-Processor OS devices) The type of co-processor, for instance `"MIC"`, `"CUDA"` or `"OpenCL"`.

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

OpenCLDeviceType, OpenCLPlatformIndex,

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

OpenCLComputeUnits, OpenCLGlobalMemorySize The number of compute units and global memory size (in kB) of a 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.

MICSerialNumber The serial number of an Intel Xeon Phi (MIC) coprocessor. hwloc may run either inside the coprocessor itself, or on the host processor. That attribute is set in both cases, so that the exact same coprocessor may be identified from both point of views, even if there are multiple nodes with multiple MICs. When running hwloc on the host, each hwloc OS device object that corresponds to a Xeon Phi gets such an attribute. When running hwloc inside a Xeon Phi coprocessor, the root object of the topology gets this attribute.

MICFamily, MICSKU, MICActiveCores, MICMemorySize The family, SKU (model), number of active cores, and memory size (in kB) of an Intel Xeon Phi (MIC) coprocessor.

DMIBoardVendor, DMIBoardName, etc. (Machine object) 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/`.

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.

Type A better type name than the usual one. This may be used to specify where Groups come from. For instance Linux S/390 *books* appear as Groups of type *Book* (see also What are these Group objects in my topology?). Block OS devices may have a Type of "Disk", "Tape", "Removable Media Device" or "Other". The Type attribute value is displayed instead of the default object type name in lstopo.

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

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 version that loaded it, but rather the first process that exported the topology to XML earlier.

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 CUS↩ TOM 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.

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.
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 or non-hwloc-valid XML contents.

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 NUMA nodes, containing a single package each, with one cache above two single-threaded cores:

```
$ lstopo -i "node:2 pack:1 cache:1 core:2 pu:1" -
Machine (2048MB)
  NUMANode L#0 (P#0 1024MB) + Package L#0 + L2 L#0 (4096KB)
    Core L#0 + PU L#0 (P#0)
    Core L#1 + PU L#1 (P#1)
  NUMANode L#1 (P#1 1024MB) + Package L#1 + 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 `machine`, `node`, `package`, `core`, `cache`, `pu`, `group`. 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). Type-specific attributes may also be given such as `L2iCache` (hwloc_obj_type_sscanf() is used for parsing the type names). Note that I/O and Misc objects are not available.

The root object does not appear in the string. A Machine object is used by default, and a System object replaces it if a Machine level is specified in the string.

Cache level depths are automatically chosen by hwloc (only a L2 first, then a L1 under it, then L3 above, then L4 etc.) unless they are specified. Memory and cache sizes are also automatically chosen 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_COMPONENTS environment variable to something such as `synthetic="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 Cache:1 Cache:2 Cache:1 Cache:1 Core:1 PU:2
```

The exported string may be passed back to hwloc for recreating another similar topology. 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, which means the `symmetric_subtree` field of the root object is set. This usually implies that I/O objects are disabled since attaching I/O busses often cause the topology to become asymmetric. Passing `--no-io` to lstopo is therefore often useful to make synthetic export work (as well as not passing any I/O topology flag before exporting with hwloc_topology_export_synthetic()).

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).

Myrinet Express **hwloc/myriexpress.h** offers interoperability with the Myrinet Express interface. It can return the list of processors near a Myrinet board managed by the MX driver. Note that if I/O device discovery is enabled, such boards may also appear as PCI objects in the topology.

Intel Xeon Phi (MIC) **hwloc/intel-mic.h** helps interoperability with Intel Xeon Phi (MIC) coprocessors by returning the list of processors near these devices. It may also return the corresponding OS device hwloc object for further information (if I/O device discovery is enabled).

AMD OpenCL **hwloc/opencl.h** enables interoperability with the OpenCL interface. Only the AMD implementation currently offers locality information. It may return the list of processors near an AMD/ATI 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_by_*` (see Modifying a loaded Topology) may modify the topology significantly by adding objects inside the tree, changing the topology depth, etc. `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.

Locating topologies `hwloc_topology_ignore*`, `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.

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, synthetic topology description, or custom building, 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

Once topology configuration functions such as hwloc_topology_set_custom() have been taken care of, the priority order of the remaining components may be changed through the HWLOC_COMPONENTS environment variable (component names must be separated by commas).

Specifying 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.

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.

Certain component names (xml and synthetic) accept an argument (e.g. xml=file.xml). These arguments behave exactly as if the corresponding string had been passed to hwloc_topology_set_xml() or hwloc_topology_set_synthetic().

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 Adding new discovery components and plugins

The types and functions cited below are declared in the **hwloc/plugins.h** header. Components are supposed to only use hwloc public headers (**hwloc.h** and anything under the include/hwloc subdirectory) and nothing from the include/private subdirectory in the source tree.

14.4.1 Basics of discovery components

Each discovery component is defined by a hwloc_disc_component structure which contains an instantiate() callback. This function is invoked when this component is actually used by a topology. It fills a new hwloc_backend structure that usually contains discover() and/or notify_new_object() callbacks taking care of the actual topology discovery.

Note

If two discovery components have the same name, only the highest priority one is actually made available. This offers a way for third-party plugins to override existing components.

14.4.2 Registering a new discovery component

Registering components to the hwloc core relies on a `hwloc_component` structure. Its `data` field points to the previously defined `hwloc_disc_component` structure while its `type` should be HWLOC_COMPONENT_TYPE_DISC. This structure should be named `hwloc_<name>_component`.

The configure script should be modified to add `<name>` to its `hwloc_components` shell variable so that the component is actually available.

Note

The symbol name of the `hwloc_component` structure is independent of the name of the discovery component mentioned in the previous section.

When the component is statically built inside the hwloc library, the symbol `hwloc_<name>_component` is added by configure to the `src/static-components.h`. The core then registers all components listed in this file.

If the new component may be built as a plugin, the configure script should also define the shell variable `hwloc_<name>_component_maybeplugin=1`. When the configure script actually enables the component as a plugin, it will set the variable `hwloc_<name>_component` to `plugin`. The build system may then use this variable to change the way the component is built. It should create a `hwloc_<name>.so` shared object. All these files are loaded in alphabetic order, and the components they contain are registered to the hwloc core.

14.5 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).

aix, darwin, freebsd, hpux, linux, netbsd, osf, 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 be built as a plugin**.

linuxpci This component can probe PCI devices on Linux without the help of external libraries such as libpciaccess. Its priority is lower than the pci component because it misses device names.

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.

custom Custom topology support (see Multi-node 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 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 / 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 uppercase translation if the prefix supplied; HWLOC_OBJ_SYSTEM becomes FOO_HWLOC_OBJ_SYSTEM. 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 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
```

16.2 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 features. It 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_THI↩ SSYSTEM 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.

16.3 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?.

16.4 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.

16.5 Why is lstopo slow?

lstopo enables most hwloc 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_COMPONENT↩ S=-opencl,cuda,nvml`.

If loading topologies is slow because the machine contains tons of processors, one should also consider using XML (see I do not want hwloc to rediscover my enormous machine topology every time I rerun a process).

16.6 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 has encountered what looks like an error 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 output from the hwloc-gather-topology script.
****************************************************************************
```

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 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. 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).

16.7 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 and PCI link speed 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. However it only applies to Intel Knights Landing Xeon Phi for now (see Why do I need hwloc-dump-hwdata for caches on Intel Knights Landing Xeon Phi?). See also `HWLOC_DUMPED_HWDATA_DIR` in Environment Variables for details about the location of dumped files.

16.8 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, the root object may

contain a distance matrix that represents the latencies between any pairs of NUMA nodes if the BIOS and/or operating system reports them.

16.9 What happens to my topology if I disable symmetric multithreading, hyper-threading, etc. ?

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)
```

16.10 How may I ignore symmetric multithreading, hyper-threading, etc. ?

First, see What happens to my topology if I disable symmetric multithreading, hyper-threading, etc. ? 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
```

16.11 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 `Type` info attribute describes why a Group was actually added (see also Custom string infos).

hwloc currently uses Groups for the following reasons:

- AMD dual-core compute units (`Type=ComputeUnit`, in the x86 backend), but these objects are usually merged with the L2 caches.
- Intel x2APIC non-core and non-package levels (in the x86 backend).
- Windows processor groups.
- IBM S/390 "Books" on Linux (`Type=Book`).
- AIX unknown hierarchy levels.
- Distance-based groups made of close objects.
- I/O parents when I/O locality does not match any existing object.

16.12 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?).

Before hwloc v2.0, hwloc_topology_ignore_type_keep_structure() and hwloc_topology_ignore_all_keep_structure() may also make topologies assymetric by removing parts of levels, especially when part of the machine is disallowed by administrator restrictions (e.g. Linux cgroups).

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. Some types, such as Caches or Groups, are annotated with a depth or level attribute (for instance L2 cache or Group1). Moreover caches have a type attribute (for instance L1i or L1d). Such attributes 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).**

16.13 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()`.

16.14 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
```

16.15 How do I handle ABI breaks and API upgrades?

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 1.10, you should use:

```
#include <hwloc.h>
#if HWLOC_API_VERSION >= 0x00010a00
...
#endif
```

The hwloc interface will be deeply modified in release 2.0 to fix several issues of the 1.x interface. The ABI will be 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
```

You should not try to remain compatible with very old releases such as 1.1.x or earlier because HWLOC_API_VERS↩ION was added in 1.0.0 and hwloc_get_api_version() came only in 1.1.1. Also do not use the old cpuset API since it was deprecated and superseded by the bitmap API in 1.1, and later removed in 1.5.

16.16 How do I build hwloc for BlueGene/Q?

IBM BlueGene/Q machines run a standard Linux on the I/O node and a custom CNK (*Compute Node Kernel*) on the compute nodes. To run on the compute node, hwloc must be cross-compiled from the I/O node 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.

16.17 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.

16.18 How do I find the local MCDRAM NUMA node on Intel Knights Landing Xeon Phi?

Intel Knights Landing Xeon Phi processors introduce 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.

The upcoming hwloc 2.0 will address this new architecture by presenting memory in an improved way. For now, starting with 1.11.2, hwloc releases use the following approximate representation:

If a cluster only contains DDR or MCDRAM but not both, that memory is available as a local NUMA node above cores as usual.

If a cluster contains both, two distinct NUMA nodes appear. They are sibling children of a Group object of type `Cluster` (or sibling children of the Package object for non-clustered processors).

The DDR memory is the local NUMA node above cores as usual. Allocating memory from one core to its local NUMA node will therefore actually allocate it on the normal memory by default.

The local high-bandwidth MCDRAM is the second NUMA node (without any Core or PU below it). It is the next sibling of the local DDR NUMA node below the same parent object. To allocate on the faster MCDRAM, one should first find the local NUMA node (the DDR memory, by looking up parent objects), and then take the next sibling to reach the local MCDRAM NUMA node (if any).

The MCDRAM NUMA nodes may also be identified thanks to the `Type` info attribute which is set to `MCDRAM`.

16.19 Why do I need hwloc-dump-hwdata for caches on Intel Knights Landing Xeon Phi?

Intel Knights Landing Xeon Phi processors may use the on-package memory (MCDRAM) as either memory or a memory-side cache (currently reported as a L3 cache by hwloc). Details about this cache (especially its size) are currently only available to privileged users.

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 configuration changed between reboots.

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

The utility is currently unneeded on non-KNL platforms.

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

16.20 How do I build for Intel Xeon Phi coprocessor?

Note

This section does not apply to standalone Intel Knights Landing Xeon Phi.

Intel Knights Corner Xeon Phi coprocessors usually runs a Linux environment but cross-compiling from the host is required. hwloc uses standard autotools options for cross-compiling. For instance, to build for a *Knights Corner (KNC)* coprocessor:

If building with `icc`:

```
./configure CC="icc -mmic" --host=x86_64-k1om-linux --build=x86_64-unknown-linux-gnu
```

If building with the Xeon Phi-specific GCC that comes with the MPSS environment, for instance `/usr/linux-k1om-4.7/bin/x86_64-k1om-linux-gcc`:

```
export PATH=$PATH:/usr/linux-k1om-4.7/bin/
./configure --host=x86_64-k1om-linux --build=x86_64-unknown-linux-gnu
```

Chapter 17

Module Index

17.1 Modules

Here is a list of all modules:

Chapter 18

Data Structure Index

18.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 19

Module Documentation

19.1 API version

Macros

- #define HWLOC_API_VERSION 0x00010b00
- #define HWLOC_COMPONENT_ABI 4

Functions

- unsigned hwloc_get_api_version (void)

19.1.1 Detailed Description

19.1.2 Macro Definition Documentation

19.1.2.1 #define HWLOC_API_VERSION 0x00010b00

Indicate at build time which hwloc API version is being used.

19.1.2.2 #define HWLOC_COMPONENT_ABI 4

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

19.1.3 Function Documentation

19.1.3.1 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.

19.2 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

19.2.1 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.

19.2.2 Typedef Documentation

19.2.2.1 typedef hwloc_const_bitmap_t hwloc_const_cpuset_t

A non-modifiable hwloc_cpuset_t.

19.2.2.2 typedef hwloc_const_bitmap_t hwloc_const_nodeset_t

A non-modifiable hwloc_nodeset_t.

19.2.2.3 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().

19.2.2.4 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 (when the whole memory is considered as a single memory bank), the nodeset may be either empty (no memory selected) or full (whole system memory selected).

See also Converting between CPU sets and node sets.

19.3 Object Types

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_SYSTEM, HWLOC_OBJ_MACHINE, HWLOC_OBJ_NUMANODE, HWLOC_OBJ_PACKAGE, HWLOC_OBJ_CACHE, HWLOC_OBJ_CORE, HWLOC_OBJ_PU, HWLOC_OBJ_GROUP, HWLOC_OBJ_MISC, HWLOC_OBJ_BRIDGE, HWLOC_OBJ_PCI_DEVICE, HWLOC_OBJ_OS_DEVICE, 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 }
- enum hwloc_compare_types_e { HWLOC_TYPE_UNORDERED }

Functions

- int hwloc_compare_types (hwloc_obj_type_t type1, hwloc_obj_type_t type2)

19.3.1 Detailed Description

19.3.2 Typedef Documentation

19.3.2.1 typedef enum hwloc_obj_bridge_type_e hwloc_obj_bridge_type_t

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

19.3.2.2 typedef enum hwloc_obj_cache_type_e hwloc_obj_cache_type_t

Cache type.

19.3.2.3 typedef enum hwloc_obj_osdev_type_e hwloc_obj_osdev_type_t

Type of a OS device.

19.3.3 Enumeration Type Documentation

19.3.3.1 enum hwloc_compare_types_e

Enumerator

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

19.3.3.2 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.

19.3.3.3 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. Only used when the HWLOC_TOPOLOGY_FLAG_↩ ICACHES topology flag is set.

19.3.3.4 enum hwloc_obj_osdev_type_e

Type of a OS device.

Enumerator

HWLOC_OBJ_OSDEV_BLOCK Operating system block device. For instance "sda" 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" Infini↩ Band 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.

19.3.3.5 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_SYSTEM Whole system (may be a cluster of machines). The whole system that is accessible to hwloc. That may comprise several machines in SSI systems like Kerrighed.

HWLOC_OBJ_MACHINE Machine. The typical root object type. A set of processors and memory with cache coherency.

HWLOC_OBJ_NUMANODE NUMA node. A set of processors around memory which the processors can directly access.

HWLOC_OBJ_PACKAGE Physical package, what goes into a socket. In the physical meaning, i.e. that you can add or remove physically.

HWLOC_OBJ_CACHE Cache. Can be L1i, L1d, L2, L3, ...

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

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 ignored when they do not bring any structure.

HWLOC_OBJ_MISC Miscellaneous objects. 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).

HWLOC_OBJ_BRIDGE Bridge. Any bridge that connects the host or an I/O bus, to another I/O bus. Bridge objects have neither CPU sets nor node sets. They are not added to the topology unless I/O discovery is enabled with hwloc_topology_set_flags().

HWLOC_OBJ_PCI_DEVICE PCI device. These objects have neither CPU sets nor node sets. They are not added to the topology unless I/O discovery is enabled with hwloc_topology_set_flags().

HWLOC_OBJ_OS_DEVICE Operating system device. These objects have neither CPU sets nor node sets. They are not added to the topology unless I/O discovery is enabled with hwloc_topology_set_flags().

HWLOC_OBJ_TYPE_MAX Sentinel value

19.3.4 Function Documentation

19.3.4.1 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.
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.

19.4 Object Structure and Attributes

Data Structures

- struct hwloc_obj_memory_s
- struct hwloc_obj_memory_s::hwloc_obj_memory_page_type_s
- struct hwloc_obj
- union hwloc_obj_attr_u
- struct hwloc_obj_attr_u::hwloc_cache_attr_s
- struct hwloc_obj_attr_u::hwloc_group_attr_s
- struct hwloc_obj_attr_u::hwloc_pcidev_attr_s
- struct hwloc_obj_attr_u::hwloc_bridge_attr_s
- struct hwloc_obj_attr_u::hwloc_osdev_attr_s
- struct hwloc_distances_s
- struct hwloc_obj_info_s

Typedefs

- typedef struct hwloc_obj * hwloc_obj_t

19.4.1 Detailed Description

19.4.2 Typedef Documentation

19.4.2.1 typedef struct hwloc_obj* hwloc_obj_t

Convenience typedef; a pointer to a struct hwloc_obj.

19.5 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)
- void hwloc_topology_check (hwloc_topology_t topology)

19.5.1 Detailed Description

19.5.2 Typedef Documentation

19.5.2.1 typedef struct hwloc_topology∗ hwloc_topology_t

Topology context.

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

19.5.3 Function Documentation

19.5.3.1 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().

19.5.3.2 void hwloc_topology_destroy (hwloc_topology_t *topology*)

Terminate and free a topology context.

Parameters

topology	is the topology to be freed

19.5.3.3 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.

19.5.3.4 int hwloc_topology_init (hwloc_topology_t * *topologyp*)

Allocate a topology context.

Parameters

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

Returns

0 on success, -1 on error.

19.5.3.5 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 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.

See also

Topology Detection Configuration and Query

19.6 Topology Detection Configuration and Query

Data Structures

- struct hwloc_topology_discovery_support
- struct hwloc_topology_cpubind_support
- struct hwloc_topology_membind_support
- struct hwloc_topology_support

Enumerations

- enum hwloc_topology_flags_e { HWLOC_TOPOLOGY_FLAG_WHOLE_SYSTEM, HWLOC_TOPOLOGY_FLAG_IS_THISSYSTEM, HWLOC_↩ TOPOLOGY_FLAG_IO_DEVICES, HWLOC_TOPOLOGY_FLAG_IO_BRIDGES, HWLOC_TOPOLOGY_FLAG_WHOLE_IO, HWLOC_TOPOLOGY_FLAG_ICACHES }

Functions

- int hwloc_topology_ignore_type (hwloc_topology_t topology, hwloc_obj_type_t type)
- int hwloc_topology_ignore_type_keep_structure (hwloc_topology_t topology, hwloc_obj_type_t type)
- int hwloc_topology_ignore_all_keep_structure (hwloc_topology_t topology)
- 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_set_pid (hwloc_topology_t restrict topology, hwloc_pid_t pid)
- int hwloc_topology_set_fsroot (hwloc_topology_t restrict topology, const char *restrict fsroot_path)
- 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_custom (hwloc_topology_t topology)
- int hwloc_topology_set_distance_matrix (hwloc_topology_t restrict topology, hwloc_obj_type_t type, unsigned nbobjs, unsigned *os_index, float *distances)
- int hwloc_topology_is_thissystem (hwloc_topology_t restrict topology)
- const struct hwloc_topology_support * hwloc_topology_get_support (hwloc_topology_t restrict topology)
- void hwloc_topology_set_userdata (hwloc_topology_t topology, const void *userdata)
- void * hwloc_topology_get_userdata (hwloc_topology_t topology)

19.6.1 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.

If none of them 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. HWLOC_FSROOT switches to reading the topology from the specified Linux filesystem root as if hwloc_topology_set_fsroot() had been called. Finally, HWLOC_THISSYSTEM enforces the return value of hwloc_↩ topology_is_thissystem().

19.6.2 Enumeration Type Documentation

19.6.2.1 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_WHOLE_SYSTEM Detect the whole system, ignore reservations and offline settings. Gather all resources, even if some were disabled by the administrator. For instance, ignore Linux Cgroup/Cpusets and gather all processors and memory nodes, and ignore the fact that some resources may be offline.

When this flag is not set, PUs 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. NUMA nodes are always added but their available memory is set to 0 when disallowed.

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_IO_DEVICES Detect PCI devices. By default, I/O devices are ignored. This flag enables I/O device detection using the pci backend. Only the common PCI devices (GPUs, NICs, block devices, ...) and host bridges (objects that connect the host objects to an I/O subsystem) will be added to the topology. Additionally it also enables MemoryModule misc objects. Uncommon devices and other bridges (such as PCI-to-PCI bridges) will be ignored.

HWLOC_TOPOLOGY_FLAG_IO_BRIDGES Detect PCI bridges. This flag should be combined with HWLOC_↩ TOPOLOGY_FLAG_IO_DEVICES to enable the detection of both common devices and of all useful bridges (bridges that have at least one device behind them).

HWLOC_TOPOLOGY_FLAG_WHOLE_IO Detect the whole PCI hierarchy. This flag enables detection of all I/O devices (even the uncommon ones such as DMA channels) and bridges (even those that have no device behind them) using the pci backend. This implies HWLOC_TOPOLOGY_FLAG_IO_DEVICES.

HWLOC_TOPOLOGY_FLAG_ICACHES Detect instruction caches. This flag enables detection of Instruction caches, instead of only Data and Unified caches.

19.6.3 Function Documentation

19.6.3.1 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().

19.6.3.2 const struct hwloc_topology_support* hwloc_topology_get_support (hwloc_topology_t restrict *topology*)

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

19.6.3.3 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().

19.6.3.4 int hwloc_topology_ignore_all_keep_structure (hwloc_topology_t *topology*)

Ignore all objects that do not bring any structure.

Ignore all objects that do not bring any structure: This is equivalent to calling hwloc_topology_ignore_type_keep_structure() for all object types.

19.6.3.5 int hwloc_topology_ignore_type (hwloc_topology_t *topology,* hwloc_obj_type_t *type*)

Ignore an object type.

Ignore all objects from the given type. The bottom-level type HWLOC_OBJ_PU may not be ignored. The top-level object of the hierarchy will never be ignored, even if this function succeeds. Group objects are always ignored if they do not bring any structure since they are designed to add structure to the topology. I/O objects may not be ignored, topology flags should be used to configure their discovery instead.

19.6.3.6 int hwloc_topology_ignore_type_keep_structure (hwloc_topology_t *topology,* hwloc_obj_type_t *type*)

Ignore an object type if it does not bring any structure.

Ignore all objects from the given type as long as they do not bring any structure: Each ignored object should have a single children or be the only child of its parent. The bottom-level type HWLOC_OBJ_PU may not be ignored. I/O objects may not be ignored, topology flags should be used to configure their discovery instead.

19.6.3.7 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).

19.6.3.8 int hwloc_topology_set_custom (hwloc_topology_t *topology*)

Prepare the topology for custom assembly.

The topology then contains a single root object. It must then be built by inserting other topologies with hwloc_custom↩ _insert_topology() or single objects with hwloc_custom_insert_group_object_by_parent(). hwloc_topology_load() must be called to finalize the new topology as usual.

Note

If nothing is inserted in the topology, hwloc_topology_load() will fail with errno set to EINVAL.
The cpuset and nodeset of the root object are NULL because these sets are meaningless when assembling multiple topologies.
On success, the custom component replaces the previously enabled component (if any), but the topology is not actually modified until hwloc_topology_load().

19.6.3.9 int hwloc_topology_set_distance_matrix (hwloc_topology_t restrict *topology*, hwloc_obj_type_t *type*, unsigned *nbobjs*, unsigned * *os_index*, float * *distances*)

Provide a distance matrix.

Provide the matrix of distances between a set of objects of the given type. `nbobjs` must be at least 2. The set may or may not contain all the existing objects of this type. The objects are specified by their OS/physical index in the `os_index` array. The `distances` matrix follows the same order. The distance from object i to object j in the i*nbobjs+j.

A single latency matrix may be defined for each type. If another distance matrix already exists for the given type, either because the user specified it or because the OS offers it, it will be replaced by the given one. If `nbobjs` is `0`, `os_index` is `NULL` and `distances` is `NULL`, the existing distance matrix for the given type is removed.

Note

Distance matrices are ignored in multi-node topologies.

19.6.3.10 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()

19.6.3.11 int hwloc_topology_set_fsroot (hwloc_topology_t restrict *topology*, const char *restrict *fsroot_path*)

Change the file-system root path when building the topology from sysfs/procfs.

On Linux system, use sysfs and procfs files as if they were mounted on the given `fsroot_path` instead of the main file-system root. Setting the environment variable HWLOC_FSROOT may also result in this behavior. Not using the main file-system root causes hwloc_topology_is_thissystem() to return 0.

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 ENOSYS on non-Linux and on Linux systems that do not support it.
-1 with the appropriate errno if `fsroot_path` cannot be used.

Note

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 Linux component replaces the previously enabled component (if any), but the topology is not actually modified until hwloc_topology_load().

19.6.3.12 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.

19.6.3.13 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 numbers describing the arity of each level. Each number may be prefixed with a type and a colon to enforce the type of a level. If only some level types are enforced, hwloc will try to choose the other types according to usual topologies, but it may fail and you may have to specify more level types manually. See also the Synthetic topologies.

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().

19.6.3.14 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.

19.6.3.15 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() 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().

19.6.3.16 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().

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().

19.7 Object levels, depths and types

Enumerations

- enum hwloc_get_type_depth_e {
  HWLOC_TYPE_DEPTH_UNKNOWN, HWLOC_TYPE_DEPTH_MULTIPLE, HWLOC_TYPE_DEPTH_BRIDGE,
  HWLOC_TYPE_DEPTH_PCI_DEVICE,
  HWLOC_TYPE_DEPTH_OS_DEVICE }

Functions

- unsigned hwloc_topology_get_depth (hwloc_topology_t restrict topology)
- int hwloc_get_type_depth (hwloc_topology_t topology, hwloc_obj_type_t type)
- 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, unsigned depth)
- unsigned hwloc_get_nbobjs_by_depth (hwloc_topology_t topology, unsigned 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, unsigned 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, unsigned 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)

19.7.1 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.

19.7.2 Enumeration Type Documentation

19.7.2.1 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.

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.

19.7.3 Function Documentation

19.7.3.1 hwloc_obj_type_t hwloc_get_depth_type (hwloc_topology_t *topology,* unsigned *depth*)

Returns the type of objects at depth `depth`.

`depth` should between 0 and hwloc_topology_get_depth()-1.

Returns

-1 if depth `depth` does not exist.

19.7.3.2 unsigned hwloc_get_nbobjs_by_depth (hwloc_topology_t *topology,* unsigned *depth*)

Returns the width of level at depth `depth`.

19.7.3.3 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.

19.7.3.4 static hwloc_obj_t hwloc_get_next_obj_by_depth (hwloc_topology_t *topology,* unsigned *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`.

19.7.3.5 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().

19.7.3.6 hwloc_obj_t hwloc_get_obj_by_depth (hwloc_topology_t *topology,* unsigned *depth,* unsigned *idx*)

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

19.7.3.7 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, `NULL` is returned and ther caller may fallback to hwloc_get_obj_by_depth().

19.7.3.8 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 typically HWLOC_OBJ_MACHINE but it could be different for complex topologies.

19.7.3.9 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 some objects of the given type exist in different levels, for instance L1 and L2 caches, or L1i and L1d caches, the function returns HWLOC_TYPE_DEPTH_MULTIPLE. See hwloc_get_cache_type_depth() in **hwloc/helper.h** to better handle this case.

If an I/O object type is given, the function returns a virtual value because I/O 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.

19.7.3.10 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`.

If some objects of the given type exist in different levels, for instance L1 and L2 caches, the function returns HWLOC_TYPE_DEPTH_MULTIPLE.

19.7.3.11 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`.

If some objects of the given type exist in different levels, for instance L1 and L2 caches, the function returns HWLOC_TYPE_DEPTH_MULTIPLE.

19.7.3.12 unsigned 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.

19.8 Converting between Object Types, Sets 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_obj_cpuset_snprintf (char *restrict str, size_t size, size_t nobj, const hwloc_obj_t *restrict objs)
- int hwloc_obj_type_sscanf (const char *string, hwloc_obj_type_t *typep, int *depthattrp, void *typeattrp, size_t typeattrsize)

19.8.1 Detailed Description

19.8.2 Function Documentation

19.8.2.1 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).

19.8.2.2 int hwloc_obj_cpuset_snprintf (char *restrict *str*, size_t *size*, size_t *nobj*, const hwloc_obj_t *restrict *objs*)

Stringify the cpuset containing a set of objects.

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).

19.8.2.3 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 `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).

19.8.2.4 int hwloc_obj_type_sscanf (const char * *string*, hwloc_obj_type_t * *typep*, int * *depthattrp*, void * *typeattrp*, size_t *typeattrsize*)

Return an object type and attributes from a type string.

Convert strings such as "Package" or "Cache" into the corresponding types. Matching is case-insensitive, and only the first letters are actually required to match.

This function is guaranteed to match any string returned by hwloc_obj_type_string() or hwloc_obj_type_snprintf().

Types that have specific attributes, for instance caches and groups, may be returned in `depthattrp` and `typeattrp`. They are ignored when these pointers are `NULL`.

For instance "L2i" or "L2iCache" would return type HWLOC_OBJ_CACHE in `typep`, 2 in `depthattrp`, and HWLOC_OBJ_CACHE_TYPE_INSTRUCTION in `typeattrp` (this last pointer should point to a hwloc_obj_cache_type_t). "Group3" would return type HWLOC_OBJ_GROUP type and 3 in `depthattrp`. Attributes that are not specified in the string (for instance "Group" without a depth, or "L2Cache" without a cache type) are set to -1.

`typeattrp` is only filled if the size specified in `typeattrsize` is large enough. It is currently only used for caches, and the required size is at least the size of hwloc_obj_cache_type_t.

Returns

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

Note

This is an extended version of the now deprecated hwloc_obj_type_of_string()

19.8.2.5 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.

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

19.9 Consulting and Adding Key-Value Info Attributes

Functions

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

19.9.1 Detailed Description

19.9.2 Function Documentation

19.9.2.1 void 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.

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().

19.9.2.2 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.

19.10 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)

19.10.1 Detailed Description

It is often useful to call hwloc_bitmap_singlify() first so that a single CPU remains in the set. This way, the process will not even migrate between different CPUs inside the given set. Some operating systems also only support that kind of binding.

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.

19.10.2 Enumeration Type Documentation

19.10.2.1 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.

19.10.3 Function Documentation

19.10.3.1 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.

19.10.3.2 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.

19.10.3.3 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`.

19.10.3.4 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`.

19.10.3.5 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`.

19.10.3.6 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

19.10.3.7 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`.

19.10.3.8 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`.

19.11 Memory binding

Enumerations

- enum hwloc_membind_policy_t {
  HWLOC_MEMBIND_DEFAULT, HWLOC_MEMBIND_FIRSTTOUCH, HWLOC_MEMBIND_BIND, HWLOC_M↩
  EMBIND_INTERLEAVE,
  HWLOC_MEMBIND_REPLICATE, HWLOC_MEMBIND_NEXTTOUCH, HWLOC_MEMBIND_MIXED }
- enum hwloc_membind_flags_t {
  HWLOC_MEMBIND_PROCESS, HWLOC_MEMBIND_THREAD, HWLOC_MEMBIND_STRICT, HWLOC_ME↩
  MBIND_MIGRATE,
  HWLOC_MEMBIND_NOCPUBIND, HWLOC_MEMBIND_BYNODESET = (1<<5) }

Functions

- int hwloc_set_membind_nodeset (hwloc_topology_t topology, hwloc_const_nodeset_t nodeset, hwloc_↩
  membind_policy_t policy, int flags)
- int hwloc_set_membind (hwloc_topology_t topology, hwloc_const_bitmap_t set, hwloc_membind_policy_t policy, int flags)
- int hwloc_get_membind_nodeset (hwloc_topology_t topology, hwloc_nodeset_t nodeset, 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_nodeset (hwloc_topology_t topology, hwloc_pid_t pid, hwloc_const_nodeset_↩
  t nodeset, 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_nodeset (hwloc_topology_t topology, hwloc_pid_t pid, hwloc_nodeset_t nodeset, 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_nodeset (hwloc_topology_t topology, const void *addr, size_t len, hwloc_const_↩
  nodeset_t nodeset, 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_nodeset (hwloc_topology_t topology, const void *addr, size_t len, hwloc_nodeset↩
  _t nodeset, 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_nodeset (hwloc_topology_t topology, size_t len, hwloc_const_nodeset_t nodeset, hwloc_membind_policy_t policy, int flags)
- 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_nodeset (hwloc_topology_t topology, size_t len, hwloc_const_↩
  nodeset_t nodeset, 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)

19.11.1 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 is available in two forms: one that takes a bitmap argument (a CPU set by default, or a NUMA memory node set if the flag HWLOC_MEMBIND_BYNODESET is specified), and another one (whose name ends with _nodeset) that always takes a NUMA memory node set. 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

19.11.2 Enumeration Type Documentation

19.11.2.1 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 multithreaded) 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. Functions whose name ends with _nodeset() take a nodeset argument. Other functions take a bitmap argument that is considered a nodeset if this flag is given, or a cpuset otherwise.

Memory binding by CPU set cannot work for CPU-less NUMA memory nodes. Binding by nodeset should therefore be preferred whenever possible.

19.11.2.2 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), or HWLOC_MEMB↩IND_BIND (AIX, HP-UX, OSF, Solaris, Windows). This policy is never returned by get membind functions when running on normal machines. It is only returned when binding hooks are empty because the topology was loaded from XML, or HWLOC_THISSYSTEM=0, etc.

HWLOC_MEMBIND_FIRSTTOUCH Allocate memory but do not immediately bind it to a specific locality. Instead, each page in the allocation is bound only when it is first touched. Pages are individually bound to the local NUMA node of the first thread that touches it. If there is not enough memory on the node, allocation may be done in the specified nodes before allocating on other nodes.

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_REPLICATE Replicate memory on the given nodes; reads from this memory will attempt to be serviced from the NUMA node local to the reading thread. Replicating can be useful when multiple threads from the specified NUMA nodes will be sharing the same read-only data. This policy can only be used with existing memory allocations (i.e., the hwloc_set_*membind*() functions); it cannot be used with functions that allocate new memory (i.e., the hwloc_alloc*() functions).

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.

19.11.3 Function Documentation

19.11.3.1 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().

19.11.3.2 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().

19.11.3.3 void* hwloc_alloc_membind_nodeset (hwloc_topology_t *topology*, size_t *len*, hwloc_const_nodeset_t *nodeset*, hwloc_membind_policy_t *policy*, int *flags*)

Allocate some memory on NUMA memory nodes specified by `nodeset`.

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.

Note

The allocated memory should be freed with hwloc_free().

19.11.3.4 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.

19.11.3.5 static void* hwloc_alloc_membind_policy_nodeset (hwloc_topology_t *topology*, size_t *len*, hwloc_const_nodeset_t *nodeset*, hwloc_membind_policy_t *policy*, int *flags*) [inline], [static]

Allocate some memory on NUMA memory nodes specified by `nodeset`.

This is similar to hwloc_alloc_membind() 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.

19.11.3.6 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().

19.11.3.7 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 `len` is 0, -1 is returned and errno is set to EINVAL.

If any other flags are specified, -1 is returned and errno is set to EINVAL.

19.11.3.8 int hwloc_get_area_membind_nodeset (hwloc_topology_t *topology*, const void * *addr*, size_t *len*, hwloc_nodeset_t *nodeset*, hwloc_membind_policy_t * *policy*, int *flags*)

Query the physical NUMA node(s) and binding policy of the memory identified by (`addr`, `len`).

This function has two output parameters: `nodeset` 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 nodeset and policy are returned in `nodeset` and `policy`, respectively.

If HWLOC_MEMBIND_STRICT is not specified, `nodeset` is set to the union of all NUMA node(s) containing pages in the address range. If all pages in the target have the same policy, it is returned in `policy`. Otherwise, `policy` is set to HWLOC_MEMBIND_MIXED.

If `len` is 0, -1 is returned and errno is set to EINVAL.

If any other flags are specified, -1 is returned and errno is set to EINVAL.

19.11.3.9 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, set is considered a nodeset. Otherwise it's a cpuset.

If `len` is 0, `set` is emptied.

Flags are currently unused.

19.11.3.10 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 de-

fault 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.

19.11.3.11 int hwloc_get_membind_nodeset (hwloc_topology_t *topology*, hwloc_nodeset_t *nodeset*, 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: `nodeset` 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 `nodeset` and `policy`.

Otherwise, if HWLOC_MEMBIND_PROCESS is specified (and HWLOC_MEMBIND_STRICT is *not* specified), `nodeset` is set to the logical OR of all threads' default nodeset. 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 nodeset and policy; they are returned in `nodeset` and `policy`, respectively.

If any other flags are specified, -1 is returned and errno is set to EINVAL.

19.11.3.12 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.

19.11.3.13 int hwloc_get_proc_membind_nodeset (hwloc_topology_t *topology*, hwloc_pid_t *pid*, hwloc_nodeset_t *nodeset*, 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: `nodeset` 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 `nodeset` and `policy`.

Otherwise, `nodeset` is set to the logical OR of all threads' default nodeset. 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 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.

19.11.3.14 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

19.11.3.15 int hwloc_set_area_membind_nodeset (hwloc_topology_t *topology*, const void * *addr*, size_t *len*, hwloc_const_nodeset_t *nodeset*, hwloc_membind_policy_t *policy*, int *flags*)

Bind the already-allocated memory identified by (addr, len) to the NUMA node(s) specified by `nodeset`.

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

19.11.3.16 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

19.11.3.17 int hwloc_set_membind_nodeset (hwloc_topology_t *topology*, hwloc_const_nodeset_t *nodeset*, 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 `nodeset`.

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.

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

19.11.3.18 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.

19.11.3.19 int hwloc_set_proc_membind_nodeset (hwloc_topology_t *topology*, hwloc_pid_t *pid*, hwloc_const_nodeset_t *nodeset*, 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 `nodeset`.

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.

19.12 Modifying a loaded Topology

Enumerations

- enum hwloc_restrict_flags_e { HWLOC_RESTRICT_FLAG_ADAPT_DISTANCES, HWLOC_RESTRICT_FLAG_ADAPT_MISC, HWLOC_RESTRICT_FLAG_ADAPT_IO }

Functions

- hwloc_obj_t hwloc_topology_insert_misc_object_by_cpuset (hwloc_topology_t topology, hwloc_const_cpuset_t cpuset, const char *name)
- hwloc_obj_t hwloc_topology_insert_misc_object_by_parent (hwloc_topology_t topology, hwloc_obj_t parent, const char *name)
- int hwloc_topology_restrict (hwloc_topology_t restrict topology, hwloc_const_cpuset_t cpuset, unsigned long flags)

19.12.1 Detailed Description

19.12.2 Enumeration Type Documentation

19.12.2.1 enum hwloc_restrict_flags_e

Flags to be given to hwloc_topology_restrict().

Enumerator

HWLOC_RESTRICT_FLAG_ADAPT_DISTANCES Adapt distance matrices according to objects being removed during restriction. If this flag is not set, distance matrices are removed.

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.

19.12.3 Function Documentation

19.12.3.1 hwloc_obj_t hwloc_topology_insert_misc_object_by_cpuset (hwloc_topology_t *topology*, hwloc_const_cpuset_t *cpuset*, const char * *name*)

Add a MISC object to the topology.

A new MISC object will be created and inserted into the topology at the position given by bitmap `cpuset`. This offers a way to add new intermediate levels to the topology hierarchy.

`cpuset` and `name` will be copied to setup the new object attributes.

Returns

the newly-created object.
`NULL` if the insertion conflicts with the existing topology tree.

Note

If `name` contains some non-printable characters, they will be dropped when exporting to XML, see hwloc_topology_export_xml().

19.12.3.2 hwloc_obj_t hwloc_topology_insert_misc_object_by_parent (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 children, without ever adding any intermediate hierarchy level. This is useful for annotating the topology without actually changing the hierarchy.

`name` will be copied to the setup the new object attributes. However, the new leaf object will not have any `cpuset`.

Returns

the newly-created object

Note

If `name` contains some non-printable characters, they will be dropped when exporting to XML, see hwloc_topology_export_xml().

19.12.3.3 int hwloc_topology_restrict (hwloc_topology_t restrict *topology*, hwloc_const_cpuset_t *cpuset*, unsigned long *flags*)

Restrict the topology to the given CPU set.

Topology `topology` is modified so as to remove all objects that are not included (or partially included) in the CPU set `cpuset`. All objects CPU and node sets are restricted accordingly.

`flags` is a OR'ed set of hwloc_restrict_flags_e.

Note

This call may not be reverted by restricting back to a larger cpuset. 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 cpuset 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.

19.13 Building Custom Topologies

Functions

- int hwloc_custom_insert_topology (hwloc_topology_t newtopology, hwloc_obj_t newparent, hwloc_topology_t oldtopology, hwloc_obj_t oldroot)
- hwloc_obj_t hwloc_custom_insert_group_object_by_parent (hwloc_topology_t topology, hwloc_obj_t parent, int groupdepth)

19.13.1 Detailed Description

A custom topology may be initialized by calling hwloc_topology_set_custom() after hwloc_topology_init(). It may then be modified by inserting objects or entire topologies. Once done assembling, hwloc_topology_load() should be invoked as usual to finalize the topology.

19.13.2 Function Documentation

19.13.2.1 hwloc_obj_t hwloc_custom_insert_group_object_by_parent (hwloc_topology_t *topology*, hwloc_obj_t *parent*, int *groupdepth*)

Insert a new group object inside a custom topology.

An object with type HWLOC_OBJ_GROUP is inserted as a new child of object `parent`.

`groupdepth` is the depth attribute to be given to the new object. It may for instance be 0 for top-level groups, 1 for their children, and so on.

The custom topology `newtopology` must have been prepared with hwloc_topology_set_custom() and not loaded with hwloc_topology_load() yet.

`parent` may be either the root of `topology` or an object that was added earlier through hwloc_custom_insert_group_object_by_parent().

Note

The cpuset and nodeset of the new group object are NULL because these sets are meaningless when assembling multiple topologies.
The cpuset and nodeset of the `parent` object are not modified.

19.13.2.2 int hwloc_custom_insert_topology (hwloc_topology_t *newtopology*, hwloc_obj_t *newparent*, hwloc_topology_t *oldtopology*, hwloc_obj_t *oldroot*)

Insert an existing topology inside a custom topology.

Duplicate the existing topology `oldtopology` inside a new custom topology `newtopology` as a leaf of object `newparent`.

If `oldroot` is not `NULL`, duplicate `oldroot` and all its children instead of the entire `oldtopology`. Passing the root object of `oldtopology` in `oldroot` is equivalent to passing `NULL`.

The custom topology `newtopology` must have been prepared with hwloc_topology_set_custom() and not loaded with hwloc_topology_load() yet.

`newparent` may be either the root of `newtopology` or an object that was added through hwloc_custom_insert_group_object_by_parent().

Note

The cpuset and nodeset of the `newparent` object are not modified based on the contents of `oldtopology`.

19.14 Exporting Topologies to XML

Functions

- int hwloc_topology_export_xml (hwloc_topology_t topology, const char *xmlpath)
- int hwloc_topology_export_xmlbuffer (hwloc_topology_t topology, char **xmlbuffer, int *buflen)
- 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))

19.14.1 Detailed Description

19.14.2 Function Documentation

19.14.2.1 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.

19.14.2.2 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.

19.14.2.3 void hwloc_free_xmlbuffer (hwloc_topology_t *topology*, char * *xmlbuffer*)

Free a buffer allocated by hwloc_topology_export_xmlbuffer()

19.14.2.4 int hwloc_topology_export_xml (hwloc_topology_t *topology*, const char * *xmlpath*)

Export the topology into an XML file.

This file may be loaded later through hwloc_topology_set_xml().

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.

19.14.2.5 int hwloc_topology_export_xmlbuffer (hwloc_topology_t *topology*, char ** *xmlbuffer*, int * *buflen*)

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().

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.

19.14.2.6 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.

19.14.2.7 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 ('\0').
This function should be called before hwloc_topology_load().
The topology-specific userdata pointer is ignored when importing from XML.

19.15 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 }

Functions

- int hwloc_topology_export_synthetic (hwloc_topology_t topology, char ∗buffer, size_t buflen, unsigned long flags)

19.15.1 Detailed Description

19.15.2 Enumeration Type Documentation

19.15.2.1 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.

19.15.3 Function Documentation

19.15.3.1 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 \0.

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 \0.
-1 if the topology could not be exported, for instance if it is not symmetric.

Note

A 1024-byte buffer should be large enough for exporting topologies in the vast majority of cases.

19.16 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, unsigned 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, unsigned 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, unsigned 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)

19.16.1 Detailed Description

19.16.2 Function Documentation

19.16.2.1 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.

Note

This function cannot work if the root object does not have a CPU set, e.g. if the topology is made of different machines.

19.16.2.2 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`.

Note

This function cannot work if the root object does not have a CPU set, e.g. if the topology is made of different machines.

19.16.2.3 static unsigned hwloc_get_nbobjs_inside_cpuset_by_depth (hwloc_topology_t *topology,* hwloc_const_cpuset_t *set,* unsigned *depth*) [inline], [static]

Return the number of objects at depth `depth` included in CPU set `set`.

Note

This function cannot work if objects at the given depth do not have CPU sets or if the topology is made of different machines.

19.16.2.4 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

This function cannot work if objects of the given type do not have CPU sets or if the topology is made of different machines.

19.16.2.5 static hwloc_obj_t hwloc_get_next_obj_inside_cpuset_by_depth (hwloc_topology_t *topology,* hwloc_const_cpuset_t *set,* unsigned *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

This function cannot work if objects at the given depth do not have CPU sets or if the topology is made of different machines.

19.16.2.6 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

This function cannot work if objects of the given type do not have CPU sets or if the topology is made of different machines.

19.16.2.7 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`.

19.16.2.8 static hwloc_obj_t hwloc_get_obj_inside_cpuset_by_depth (hwloc_topology_t *topology,* hwloc_const_cpuset_t *set,* unsigned *depth,* unsigned *idx*) [inline], [static]

Return the (logically) `idx` -th object at depth `depth` included in CPU set `set`.

Note

This function cannot work if objects at the given depth do not have CPU sets or if the topology is made of different machines.

19.16.2.9 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

This function cannot work if objects of the given type do not have CPU sets or if the topology is made of different machines.

19.17 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, unsigned 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)

19.17.1 Detailed Description

19.17.2 Function Documentation

19.17.2.1 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.

19.17.2.2 static hwloc_obj_t hwloc_get_next_obj_covering_cpuset_by_depth (hwloc_topology_t *topology,* hwloc_const_cpuset_t *set,* unsigned *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 or if the topology is made of different machines.

19.17.2.3 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 or if the topology is made of different machines.

19.17.2.4 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.

Note

This function cannot work if the root object does not have a CPU set, e.g. if the topology is made of different machines.

19.18 Looking at Ancestor and Child Objects

Functions

- static hwloc_obj_t hwloc_get_ancestor_obj_by_depth (hwloc_topology_t topology, unsigned 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)

19.18.1 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.

19.18.2 Function Documentation

19.18.2.1 static hwloc_obj_t hwloc_get_ancestor_obj_by_depth (hwloc_topology_t *topology,* unsigned *depth,* hwloc_obj_t *obj*) [inline], [static]

Returns the ancestor object of `obj` at depth `depth`.

19.18.2.2 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`.

19.18.2.3 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`.

19.18.2.4 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.

If `prev` is `NULL`, return the first child.

19.18.2.5 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 assumes that both `obj` and `subtree_root` have a `cpuset`.

19.19 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)

19.19.1 Detailed Description

19.19.2 Function Documentation

19.19.2.1 static hwloc_obj_t hwloc_get_cache_covering_cpuset (hwloc_topology_t *topology,* hwloc_const_cpuset_t *set*) [inline], [static]

Get the first cache covering a cpuset `set`.

Returns

`NULL` if no cache matches.

Note

This function cannot work if the root object does not have a CPU set, e.g. if the topology is made of different machines.

19.19.2.2 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 depth and type.

Return the depth of the topology level that contains cache objects whose attributes match `cachedepth` and `cachetype`. This function intends to disambiguate the case where hwloc_get_type_depth() returns HWLOC_TYPE_DEPTH_MULTIPLE.

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.

19.19.2.3 static hwloc_obj_t hwloc_get_shared_cache_covering_obj (hwloc_topology_t *topology,* hwloc_obj_t *obj*) [inline], [static]

Get the first cache shared between an object and somebody else.

Returns

`NULL` if no cache matches or if an invalid object is given.

19.20 Finding objects, miscellaneous helpers

Functions

- 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)

19.20.1 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.

19.20.2 Function Documentation

19.20.2.1 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.

19.20.2.2 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_nodeset()), one may iterate over the bits of the resulting nodeset with hwloc_bitmap_foreach_begin(), and find the corresponding NUMA nodes with this function.

19.20.2.3 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.

19.20.2.4 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.

19.20.2.5 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.

19.21 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, unsigned until, unsigned long flags)

19.21.1 Detailed Description

19.21.2 Enumeration Type Documentation

19.21.2.1 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.

19.21.3 Function Documentation

19.21.3.1 static int hwloc_distrib (hwloc_topology_t *topology,* hwloc_obj_t * *roots,* unsigned *n_roots,* hwloc_cpuset_t * *set,* unsigned *n,* unsigned *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.

19.22 CPU and node sets of entire topologies

Functions

- static hwloc_const_cpuset_t hwloc_topology_get_complete_cpuset (hwloc_topology_t topology)
- static hwloc_const_cpuset_t hwloc_topology_get_topology_cpuset (hwloc_topology_t topology)
- static hwloc_const_cpuset_t hwloc_topology_get_online_cpuset (hwloc_topology_t topology)
- static hwloc_const_cpuset_t hwloc_topology_get_allowed_cpuset (hwloc_topology_t topology)
- static hwloc_const_nodeset_t hwloc_topology_get_complete_nodeset (hwloc_topology_t topology)
- static hwloc_const_nodeset_t hwloc_topology_get_topology_nodeset (hwloc_topology_t topology)
- static hwloc_const_nodeset_t hwloc_topology_get_allowed_nodeset (hwloc_topology_t topology)

19.22.1 Detailed Description

19.22.2 Function Documentation

19.22.2.1 static hwloc_const_cpuset_t hwloc_topology_get_allowed_cpuset (hwloc_topology_t *topology*) [inline], [static]

Get allowed CPU set.

Returns

the CPU set of allowed logical processors of the system. If the topology is the result of a combination of several systems, NULL is returned.

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.

19.22.2.2 static hwloc_const_nodeset_t hwloc_topology_get_allowed_nodeset (hwloc_topology_t *topology*) [inline], [static]

Get allowed node set.

Returns

the node set of allowed memory of the system. If the topology is the result of a combination of several systems, NULL is returned.

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.

19.22.2.3 static hwloc_const_cpuset_t hwloc_topology_get_complete_cpuset (hwloc_topology_t *topology*) [inline], [static]

Get complete CPU set.

Returns

the complete CPU set of logical processors of the system. If the topology is the result of a combination of several systems, NULL is returned.

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.

19.22.2.4 static hwloc_const_nodeset_t hwloc_topology_get_complete_nodeset (hwloc_topology_t *topology*) [inline], [static]

Get complete node set.

Returns

the complete node set of memory of the system. If the topology is the result of a combination of several systems, NULL is returned.

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.

19.22.2.5 static hwloc_const_cpuset_t hwloc_topology_get_online_cpuset (hwloc_topology_t *topology*) [inline], [static]

Get online CPU set.

Returns

the CPU set of online logical processors of the system. If the topology is the result of a combination of several systems, NULL is returned.

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.

19.22.2.6 static hwloc_const_cpuset_t hwloc_topology_get_topology_cpuset (hwloc_topology_t *topology*) [inline], [static]

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. If the topology is the result of a combination of several systems, NULL is returned.

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.

19.22.2.7 static hwloc_const_nodeset_t hwloc_topology_get_topology_nodeset (hwloc_topology_t *topology*) [inline], [static]

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. If the topology is the result of a combination of several systems, NULL is returned.

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.

19.23 Converting between CPU sets and node sets

Functions

- static void hwloc_cpuset_to_nodeset (hwloc_topology_t topology, hwloc_const_cpuset_t _cpuset, hwloc_↩ nodeset_t nodeset)
- static void hwloc_cpuset_to_nodeset_strict (struct hwloc_topology *topology, hwloc_const_cpuset_t _cpuset, hwloc_nodeset_t nodeset)
- static void hwloc_cpuset_from_nodeset (hwloc_topology_t topology, hwloc_cpuset_t _cpuset, hwloc_const_↩ nodeset_t nodeset)
- static void hwloc_cpuset_from_nodeset_strict (struct hwloc_topology *topology, hwloc_cpuset_t _cpuset, hwloc↩ _const_nodeset_t nodeset)

19.23.1 Detailed Description

There are two semantics for converting cpusets to nodesets depending on how non-NUMA machines are handled.

When manipulating nodesets for memory binding, non-NUMA machines should be considered as having a single NUMA node. The standard conversion routines below should be used so that marking the first bit of the nodeset means that memory should be bound to a non-NUMA whole machine.

When manipulating nodesets as an actual list of NUMA nodes without any need to handle memory binding on non-N↩ UMA machines, the strict conversion routines may be used instead.

19.23.2 Function Documentation

19.23.2.1 static void 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 and handle non-NUMA cases.

If the topology contains no NUMA nodes, the machine is considered as a single memory node, and the following behavior is used: If `nodeset` is empty, `cpuset` will be emptied as well. Otherwise `cpuset` will be entirely filled. This is useful for manipulating memory binding sets.

19.23.2.2 static void hwloc_cpuset_from_nodeset_strict (struct hwloc_topology * *topology*, hwloc_cpuset_t *_cpuset*, hwloc_const_nodeset_t *nodeset*) [inline], [static]

Convert a NUMA node set into a CPU set without handling non-NUMA cases.

This is the strict variant of hwloc_cpuset_from_nodeset(). It does not fix non-NUMA cases. If the topology contains some NUMA nodes, behave exactly the same. However, if the topology contains no NUMA nodes, return an empty cpuset.

19.23.2.3 static void 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 and handle non-NUMA cases.

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.

If the topology contains no NUMA nodes, the machine is considered as a single memory node, and the following behavior is used: If `cpuset` is empty, `nodeset` will be emptied as well. Otherwise `nodeset` will be entirely filled.

19.23.2.4 static void hwloc_cpuset_to_nodeset_strict (struct hwloc_topology * *topology,* hwloc_const_cpuset_t *_cpuset,* hwloc_nodeset_t *nodeset*) [inline], [static]

Convert a CPU set into a NUMA node set without handling non-NUMA cases.

This is the strict variant of hwloc_cpuset_to_nodeset(). It does not fix non-NUMA cases. If the topology contains some NUMA nodes, behave exactly the same. However, if the topology contains no NUMA nodes, return an empty nodeset.

19.24 Manipulating Distances

Functions

- static const struct hwloc_distances_s * hwloc_get_whole_distance_matrix_by_depth (hwloc_topology_t topology, unsigned depth)
- static const struct hwloc_distances_s * hwloc_get_whole_distance_matrix_by_type (hwloc_topology_t topology, hwloc_obj_type_t type)
- static const struct hwloc_distances_s * hwloc_get_distance_matrix_covering_obj_by_depth (hwloc_topology_t topology, hwloc_↩ obj_t obj, unsigned depth, unsigned *firstp)
- static int hwloc_get_latency (hwloc_topology_t topology, hwloc_obj_t obj1, hwloc_obj_t obj2, float *latency, float *reverse_latency)

19.24.1 Detailed Description

19.24.2 Function Documentation

19.24.2.1 static const struct hwloc_distances_s* hwloc_get_distance_matrix_covering_obj_by_depth (hwloc_topology_t *topology,* hwloc_obj_t *obj,* unsigned *depth,* unsigned * *firstp*) [static]

Get distances for the given depth and covering some objects.

Return a distance matrix that describes depth `depth` and covers at least object `obj` and all its children.

When looking for the distance between some objects, a common ancestor should be passed in `obj`.

`firstp` is set to logical index of the first object described by the matrix.

The returned structure belongs to the hwloc library. The caller should not modify or free it.

19.24.2.2 static int hwloc_get_latency (hwloc_topology_t *topology,* hwloc_obj_t *obj1,* hwloc_obj_t *obj2,* float * *latency,* float * *reverse_latency*) [inline], [static]

Get the latency in both directions between two objects.

Look at ancestor objects from the bottom to the top until one of them contains a distance matrix that matches the objects exactly.

`latency` gets the value from object `obj1` to `obj2`, while `reverse_latency` gets the reverse-direction value, which may be different on some architectures.

Returns

-1 if no ancestor contains a matching latency matrix.

19.24.2.3 static const struct hwloc_distances_s* hwloc_get_whole_distance_matrix_by_depth (hwloc_topology_t *topology,* unsigned *depth*) [static]

Get the distances between all objects at the given depth.

Returns

a distances structure containing a matrix with all distances between all objects at the given depth.

Slot i+nbobjs∗j contains the distance from the object of logical index i the object of logical index j.

Note

This function only returns matrices covering the whole topology, without any unknown distance value. Those matrices are available in top-level object of the hierarchy. Matrices of lower objects are not reported here since they cover only part of the machine.

The returned structure belongs to the hwloc library. The caller should not modify or free it.

Returns

NULL if no such distance matrix exists.

19.24.2.4 static const struct hwloc_distances_s∗ hwloc_get_whole_distance_matrix_by_type (hwloc_topology_t *topology*, hwloc_obj_type_t *type*) [static]

Get the distances between all objects of a given type.

Returns

a distances structure containing a matrix with all distances between all objects of the given type.

Slot i+nbobjs∗j contains the distance from the object of logical index i the object of logical index j.

Note

This function only returns matrices covering the whole topology, without any unknown distance value. Those matrices are available in top-level object of the hierarchy. Matrices of lower objects are not reported here since they cover only part of the machine.

The returned structure belongs to the hwloc library. The caller should not modify or free it.

Returns

NULL if no such distance matrix exists.

19.25 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)
- static hwloc_obj_t hwloc_get_hostbridge_by_pcibus (hwloc_topology_t topology, unsigned domain, unsigned bus)

19.25.1 Detailed Description

19.25.2 Function Documentation

19.25.2.1 static int hwloc_bridge_covers_pcibus (hwloc_obj_t *bridge,* unsigned *domain,* unsigned *bus*) [inline], [static]

19.25.2.2 static hwloc_obj_t hwloc_get_hostbridge_by_pcibus (hwloc_topology_t *topology,* unsigned *domain,* unsigned *bus*) [inline], [static]

Find the hostbridge that covers the given PCI bus.

This is useful for finding the locality of a bus because it is the hostbridge parent cpuset.

19.25.2.3 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`.

19.25.2.4 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`.

19.25.2.5 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`.

19.25.2.6 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 regular object may then be used for binding because its locality is the same as `ioobj`.

19.25.2.7 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.

19.25.2.8 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.

19.26 The bitmap API

Macros

- #define hwloc_bitmap_foreach_begin(id, bitmap)
- #define hwloc_bitmap_foreach_end()

Typedefs

- typedef struct hwloc_bitmap_s * hwloc_bitmap_t
- typedef const 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)
- void 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)
- void hwloc_bitmap_only (hwloc_bitmap_t bitmap, unsigned id)
- void hwloc_bitmap_allbut (hwloc_bitmap_t bitmap, unsigned id)
- void hwloc_bitmap_from_ulong (hwloc_bitmap_t bitmap, unsigned long mask)
- void hwloc_bitmap_from_ith_ulong (hwloc_bitmap_t bitmap, unsigned i, unsigned long mask)
- void hwloc_bitmap_set (hwloc_bitmap_t bitmap, unsigned id)
- void hwloc_bitmap_set_range (hwloc_bitmap_t bitmap, unsigned begin, int end)
- void hwloc_bitmap_set_ith_ulong (hwloc_bitmap_t bitmap, unsigned i, unsigned long mask)
- void hwloc_bitmap_clr (hwloc_bitmap_t bitmap, unsigned id)
- void hwloc_bitmap_clr_range (hwloc_bitmap_t bitmap, unsigned begin, int end)
- void 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_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)

- void hwloc_bitmap_or (hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)
- void hwloc_bitmap_and (hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)
- void hwloc_bitmap_andnot (hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)
- void hwloc_bitmap_xor (hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)
- void 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)

19.26.1 Detailed Description

The hwloc_bitmap_t type represents a set of objects, typically OS processors – which may actually be hardware threads (represented by hwloc_cpuset_t, which is a typedef for hwloc_bitmap_t) – or memory nodes (represented by hwloc_nodeset_t, which is also a typedef for hwloc_bitmap_t).

Both CPU and node sets are always indexed by OS physical number.

Note

CPU sets and nodesets are described in Object Sets (hwloc_cpuset_t and hwloc_nodeset_t).

A bitmap may be of infinite size.

Note

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.

19.26.2 Macro Definition Documentation

19.26.2.1 #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 infinite.

19.26.2.2 #define hwloc_bitmap_foreach_end( )

End of loop macro iterating on a bitmap.

Needs a terminating ';'.

See also

hwloc_bitmap_foreach_begin()

19.26.3 Typedef Documentation

19.26.3.1 typedef struct hwloc_bitmap_s* hwloc_bitmap_t

Set of bits represented as an opaque pointer to an internal bitmap.

19.26.3.2 typedef const struct hwloc_bitmap_s* hwloc_const_bitmap_t

a non-modifiable hwloc_bitmap_t

19.26.4 Function Documentation

19.26.4.1 void hwloc_bitmap_allbut (hwloc_bitmap_t *bitmap,* unsigned *id*)

Fill the bitmap `and` clear the index `id`.

19.26.4.2 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().

19.26.4.3 hwloc_bitmap_t hwloc_bitmap_alloc_full (void)

Allocate a new full bitmap.

19.26.4.4 void 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`

19.26.4.5 void 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`

19.26.4.6 int hwloc_bitmap_asprintf (char ** *strp,* hwloc_const_bitmap_t *bitmap*)

Stringify a bitmap into a newly allocated string.

19.26.4.7 void hwloc_bitmap_clr (hwloc_bitmap_t *bitmap,* unsigned *id*)

Remove index `id` from bitmap `bitmap`.

19.26.4.8 void 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.

19.26.4.9 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.

Note

This is different from the non-existing hwloc_bitmap_compare_last() which would only compare the highest index of each bitmap.

19.26.4.10 int hwloc_bitmap_compare_first (hwloc_const_bitmap_t *bitmap1,* hwloc_const_bitmap_t *bitmap2*)

Compare bitmaps `bitmap1` and `bitmap2` using their lowest index.

Smaller least significant bit is smaller. The empty bitmap is considered higher than anything.

19.26.4.11 void hwloc_bitmap_copy (hwloc_bitmap_t *dst,* hwloc_const_bitmap_t *src*)

Copy the contents of bitmap `src` into the already allocated bitmap `dst`.

19.26.4.12 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.

19.26.4.13 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)

19.26.4.14 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.

19.26.4.15 void hwloc_bitmap_free (hwloc_bitmap_t *bitmap*)

Free bitmap `bitmap`.

If `bitmap` is `NULL`, no operation is performed.

19.26.4.16 void 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.

19.26.4.17 void hwloc_bitmap_from_ulong (hwloc_bitmap_t *bitmap,* unsigned long *mask*)

Setup bitmap `bitmap` from unsigned long `mask`.

19.26.4.18 int hwloc_bitmap_intersects (hwloc_const_bitmap_t *bitmap1,* hwloc_const_bitmap_t *bitmap2*)

Test whether bitmaps `bitmap1` and `bitmap2` intersects.

19.26.4.19 int hwloc_bitmap_isequal (hwloc_const_bitmap_t *bitmap1,* hwloc_const_bitmap_t *bitmap2*)

Test whether bitmap `bitmap1` is equal to bitmap `bitmap2`.

19.26.4.20 int hwloc_bitmap_isfull (hwloc_const_bitmap_t *bitmap*)

Test whether bitmap `bitmap` is completely full.

19.26.4.21 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`.

19.26.4.22 int hwloc_bitmap_isset (hwloc_const_bitmap_t *bitmap,* unsigned *id*)

Test whether index `id` is part of bitmap `bitmap`.

19.26.4.23 int hwloc_bitmap_iszero (hwloc_const_bitmap_t *bitmap*)

Test whether bitmap `bitmap` is empty.

19.26.4.24 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 bitmap, or if the index bitmap is infinite.

19.26.4.25 int hwloc_bitmap_list_asprintf (char ** *strp,* hwloc_const_bitmap_t *bitmap*)

Stringify a bitmap into a newly allocated list string.

19.26.4.26 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 a ending indexes if the bitmap is infinite.

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).

19.26.4.27 int hwloc_bitmap_list_sscanf (hwloc_bitmap_t *bitmap,* const char *restrict *string*)

Parse a list string and stores it in bitmap `bitmap`.

19.26.4.28 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 bitmap.

19.26.4.29 void 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`

19.26.4.30 void hwloc_bitmap_only (hwloc_bitmap_t *bitmap,* unsigned *id*)

Empty the bitmap `bitmap` and add bit `id`.

19.26.4.31 void 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`

19.26.4.32 void hwloc_bitmap_set (hwloc_bitmap_t *bitmap,* unsigned *id*)

Add index `id` in bitmap `bitmap`.

19.26.4.33 void 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`.

19.26.4.34 void 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.

19.26.4.35 void 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.

19.26.4.36 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).

19.26.4.37 int hwloc_bitmap_sscanf (hwloc_bitmap_t *bitmap,* const char *restrict *string*)

Parse a bitmap string and stores it in bitmap `bitmap`.

19.26.4.38 int hwloc_bitmap_taskset_asprintf (char ** *strp,* hwloc_const_bitmap_t *bitmap*)

Stringify a bitmap into a newly allocated taskset-specific string.

19.26.4.39 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).

19.26.4.40 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`.

19.26.4.41 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.

19.26.4.42 unsigned long hwloc_bitmap_to_ulong (hwloc_const_bitmap_t *bitmap*)

Convert the beginning part of bitmap `bitmap` into unsigned long `mask`.

19.26.4.43 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.

19.26.4.44 void 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`

19.26.4.45 void hwloc_bitmap_zero (hwloc_bitmap_t *bitmap*)

Empty the bitmap `bitmap`.

19.27 Topology differences

Data Structures

- union hwloc_topology_diff_obj_attr_u
- struct hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_generic_s
- struct hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_uint64_s
- struct hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_string_s
- union hwloc_topology_diff_u
- struct hwloc_topology_diff_u::hwloc_topology_diff_generic_s
- struct hwloc_topology_diff_u::hwloc_topology_diff_obj_attr_s
- struct hwloc_topology_diff_u::hwloc_topology_diff_too_complex_s

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_t topology, hwloc_topology_diff_t diff)
- int hwloc_topology_diff_load_xml (hwloc_topology_t topology, const char *xmlpath, hwloc_topology_diff_t *diff, char **refname)
- int hwloc_topology_diff_export_xml (hwloc_topology_t topology, hwloc_topology_diff_t diff, const char *refname, const char *xmlpath)
- int hwloc_topology_diff_load_xmlbuffer (hwloc_topology_t topology, const char *xmlbuffer, int buflen, hwloc_topology_diff_t *diff, char **refname)
- int hwloc_topology_diff_export_xmlbuffer (hwloc_topology_t topology, hwloc_topology_diff_t diff, const char *refname, char **xmlbuffer, int *buflen)

19.27.1 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.

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.

19.27.2 Typedef Documentation

19.27.2.1 typedef enum hwloc_topology_diff_obj_attr_type_e hwloc_topology_diff_obj_attr_type_t

Type of one object attribute difference.

19.27.2.2 typedef union hwloc_topology_diff_u * hwloc_topology_diff_t

One element of a difference list between two topologies.

19.27.2.3 typedef enum hwloc_topology_diff_type_e hwloc_topology_diff_type_t

Type of one element of a difference list.

19.27.3 Enumeration Type Documentation

19.27.3.1 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.

19.27.3.2 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.

19.27.3.3 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.

19.27.4 Function Documentation

19.27.4.1 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.

19.27.4.2 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).

19.27.4.3 int hwloc_topology_diff_destroy (hwloc_topology_t *topology*, hwloc_topology_diff_t *diff*)

Destroy a list of topology differences.

Note

The topology parameter must be a valid topology but it is not required that it is related to diff.

19.27.4.4 int hwloc_topology_diff_export_xml (hwloc_topology_t *topology*, 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.

Note

The topology parameter must be a valid topology but it is not required that it is related to diff.

19.27.4.5 int hwloc_topology_diff_export_xmlbuffer (hwloc_topology_t *topology*, 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.

Note

The XML buffer should later be freed with hwloc_free_xmlbuffer().
The topology parameter must be a valid topology but it is not required that it is related to diff.

19.27.4.6 int hwloc_topology_diff_load_xml (hwloc_topology_t *topology*, 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 `topology` parameter must be a valid topology but it is not required that it is related to `diff`.
the pointer returned in refname should later be freed by the caller.

19.27.4.7 int hwloc_topology_diff_load_xmlbuffer (hwloc_topology_t *topology*, 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 `topology` parameter must be a valid topology but it is not required that it is related to `diff`.
the pointer returned in refname should later be freed by the caller.

19.28 Components and Plugins: Discovery components

Data Structures

- struct hwloc_disc_component

Typedefs

- typedef enum hwloc_disc_component_type_e hwloc_disc_component_type_t

Enumerations

- enum hwloc_disc_component_type_e { HWLOC_DISC_COMPONENT_TYPE_CPU, HWLOC_DISC_COMPONENT_TYPE_GLOBAL, HWLOC_DISC_COMPONENT_TYPE_MISC }

19.28.1 Detailed Description

19.28.2 Typedef Documentation

19.28.2.1 typedef enum hwloc_disc_component_type_e hwloc_disc_component_type_t

Discovery component type.

19.28.3 Enumeration Type Documentation

19.28.3.1 enum hwloc_disc_component_type_e

Discovery component type.

Enumerator

HWLOC_DISC_COMPONENT_TYPE_CPU CPU-only discovery through the OS, or generic no-OS support.

HWLOC_DISC_COMPONENT_TYPE_GLOBAL xml, synthetic or custom, platform-specific components such as bgq. Anything the discovers CPU and everything else. No misc backend is expected to complement a global component.

HWLOC_DISC_COMPONENT_TYPE_MISC OpenCL, Cuda, etc.

19.29 Components and Plugins: Discovery backends

Data Structures

- struct hwloc_backend

Enumerations

- enum hwloc_backend_flag_e { HWLOC_BACKEND_FLAG_NEED_LEVELS }

Functions

- struct hwloc_backend * hwloc_backend_alloc (struct hwloc_disc_component *component)
- int hwloc_backend_enable (struct hwloc_topology *topology, struct hwloc_backend *backend)
- int hwloc_backends_get_obj_cpuset (struct hwloc_backend *caller, struct hwloc_obj *obj, hwloc_bitmap_t cpuset)
- int hwloc_backends_notify_new_object (struct hwloc_backend *caller, struct hwloc_obj *obj)

19.29.1 Detailed Description

19.29.2 Enumeration Type Documentation

19.29.2.1 enum hwloc_backend_flag_e

Backend flags.

Enumerator

HWLOC_BACKEND_FLAG_NEED_LEVELS Levels should be reconnected before this backend discover() is used.

19.29.3 Function Documentation

19.29.3.1 struct hwloc_backend* hwloc_backend_alloc (struct hwloc_disc_component * *component*)

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().

19.29.3.2 int hwloc_backend_enable (struct hwloc_topology * *topology*, struct hwloc_backend * *backend*)

Enable a previously allocated and setup backend.

19.29.3.3 int hwloc_backends_get_obj_cpuset (struct hwloc_backend * *caller*, struct hwloc_obj * *obj*, hwloc_bitmap_t *cpuset*)

Used by backends discovery callbacks to request locality information from others.

Traverse the list of enabled backends until one has a get_obj_cpuset() method, and call it.

19.29.3.4 int hwloc_backends_notify_new_object (struct hwloc_backend * *caller,* struct hwloc_obj * *obj*)

Used by backends discovery callbacks to notify other backends of new objects.

Traverse the list of enabled backends (all but caller) and invoke their notify_new_object() method to notify them that a new object just got added to the topology.

Currently only used for notifying of new PCI device objects.

19.30 Components and Plugins: Generic components

Data Structures

- struct hwloc_component

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 }

19.30.1 Detailed Description

19.30.2 Typedef Documentation

19.30.2.1 typedef enum hwloc_component_type_e hwloc_component_type_t

Generic component type.

19.30.3 Enumeration Type Documentation

19.30.3.1 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.

19.31 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 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)
- static struct hwloc_obj ∗ hwloc_alloc_setup_object (hwloc_obj_type_t type, signed os_index)
- int hwloc_fill_object_sets (hwloc_obj_t obj)
- static int hwloc_plugin_check_namespace (const char ∗pluginname, const char ∗symbol)

19.31.1 Detailed Description

19.31.2 Typedef Documentation

19.31.2.1 typedef void(∗ hwloc_report_error_t)(const char ∗msg, int line)

Type of error callbacks during object insertion.

19.31.3 Function Documentation

19.31.3.1 struct hwloc_obj∗ hwloc__insert_object_by_cpuset (struct hwloc_topology ∗ *topology,* hwloc_obj_t *obj,* hwloc_report_error_t *report_error*)

Add an object to the topology and specify which error callback to use.

Aside from the error callback selection, this function is identical to hwloc_insert_object_by_cpuset()

19.31.3.2 static struct hwloc_obj∗ hwloc_alloc_setup_object (hwloc_obj_type_t *type,* signed *os_index*) [static]

Allocate and initialize an object of the given type and physical index.

19.31.3.3 int hwloc_fill_object_sets (hwloc_obj_t *obj*)

Setup object cpusets/nodesets by OR'ing its children.

Used when adding an object late in the topology, after propagating sets up and down. The caller should use this after inserting by cpuset (which means the cpusets is already OK). Typical case: PCI backend adding a hostbridge parent.

19.31.3.4 int hwloc_hide_errors (void)

Check whether insertion errors are hidden.

19.31.3.5 struct hwloc_obj* hwloc_insert_object_by_cpuset (struct hwloc_topology * *topology,* hwloc_obj_t *obj*)

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.

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.

19.31.3.6 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 children are inserted in order.

The given object may have children.

Remember to call topology_connect() afterwards to fix handy pointers.

19.31.3.7 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.

19.31.3.8 void hwloc_report_os_error (const char * *msg,* int *line*)

Report an insertion error from a backend.

19.32 Components and Plugins: PCI functions to be used by components

Functions

- int hwloc_insert_pci_device_list (struct hwloc_backend *backend, struct hwloc_obj *first_obj)
- unsigned hwloc_pci_find_cap (const unsigned char *config, unsigned cap)
- int hwloc_pci_find_linkspeed (const unsigned char *config, unsigned offset, float *linkspeed)
- int hwloc_pci_prepare_bridge (hwloc_obj_t obj, const unsigned char *config)

19.32.1 Detailed Description

19.32.2 Function Documentation

19.32.2.1 int hwloc_insert_pci_device_list (struct hwloc_backend * *backend*, struct hwloc_obj * *first_obj*)

Insert a list of PCI devices and bridges in the backend topology.

Insert a list of objects (either PCI device or bridges) starting at first_obj (linked by next_sibling in the topology, and ending with NULL). Objects are placed under the right bridges, and the remaining upstream bridges are then inserted in the topology by calling the get_obj_cpuset() callback to find their locality.

19.32.2.2 unsigned hwloc_pci_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.

19.32.2.3 int hwloc_pci_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.

19.32.2.4 int hwloc_pci_prepare_bridge (hwloc_obj_t *obj*, const unsigned char * *config*)

Modify the PCI device object into a bridge and fill its attribute if a bridge is found in the PCI config space.

This function requires 64 bytes of common configuration header at the beginning of config.

Returns -1 and destroys /p obj if bridge fields are invalid.

19.33 Linux-specific helpers

Functions

- int hwloc_linux_parse_cpumap_file (FILE *file, hwloc_cpuset_t set)
- 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)

19.33.1 Detailed Description

This includes helpers for manipulating Linux kernel cpumap files, and hwloc equivalents of the Linux sched_setaffinity and sched_getaffinity system calls.

19.33.2 Function Documentation

19.33.2.1 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.

19.33.2.2 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.

19.33.2.3 int hwloc_linux_parse_cpumap_file (FILE * *file*, hwloc_cpuset_t *set*)

Convert a linux kernel cpumap file `file` into hwloc CPU set.

Might be used when reading CPU set from sysfs attributes such as topology and caches for processors, or local_cpus for devices.

19.33.2.4 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.

19.34 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)

19.34.1 Detailed Description

This interface helps converting between Linux libnuma unsigned long masks and hwloc cpusets and nodesets.

It also offers a consistent behavior on non-NUMA machines or non-NUMA-aware kernels by assuming that the machines have a single NUMA node.

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.

19.34.2 Function Documentation

19.34.2.1 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).

19.34.2.2 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.

19.34.2.3 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).

19.34.2.4 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.

19.35 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)

19.35.1 Detailed Description

This interface helps converting between Linux libnuma bitmasks and hwloc cpusets and nodesets.

It also offers a consistent behavior on non-NUMA machines or non-NUMA-aware kernels by assuming that the machines have a single NUMA node.

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.

19.35.2 Function Documentation

19.35.2.1 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.

19.35.2.2 static struct bitmask * hwloc_cpuset_to_linux_libnuma_bitmask (hwloc_topology_t *topology,* hwloc_const_cpuset_t *cpuset*) [static]

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.

19.35.2.3 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.

19.35.2.4 static struct bitmask * hwloc_nodeset_to_linux_libnuma_bitmask (hwloc_topology_t *topology*, hwloc_const_nodeset_t *nodeset*) [static]

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.

19.36 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)

19.36.1 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.

19.36.2 Function Documentation

19.36.2.1 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

19.36.2.2 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

19.37 Interoperability with OpenCL

Functions

- 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)

19.37.1 Detailed Description

This interface offers ways to retrieve topology information about OpenCL devices.

Only the AMD OpenCL interface currently offers useful locality information about its devices.

19.37.2 Function Documentation

19.37.2.1 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 OpenCL implementation; other systems will simply get a full cpuset.

19.37.2.2 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 `device`.

Return the hwloc OS device object that describes the given OpenCL device `device`. Return NULL if there is none.

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

The corresponding hwloc PCI device may be found by looking at the result parent pointer.

19.37.2.3 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.

19.38 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)

19.38.1 Detailed Description

This interface offers ways to retrieve topology information about CUDA devices when using the CUDA Driver API.

19.38.2 Function Documentation

19.38.2.1 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.

19.38.2.2 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 NVML 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

The corresponding hwloc PCI device may be found by looking at the result parent pointer.

19.38.2.3 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.
This function is identical to hwloc_cudart_get_device_osdev_by_index().

19.38.2.4 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.

19.38.2.5 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.

19.39 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)

19.39.1 Detailed Description

This interface offers ways to retrieve topology information about CUDA devices when using the CUDA Runtime API.

19.39.2 Function Documentation

19.39.2.1 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.

19.39.2.2 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.
This function is identical to hwloc_cuda_get_device_osdev_by_index().

19.39.2.3 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.

19.39.2.4 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.

19.40 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)

19.40.1 Detailed Description

This interface offers ways to retrieve topology information about devices managed by the NVIDIA Management Library (NVML).

19.40.2 Function Documentation

19.40.2.1 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.

19.40.2.2 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.

19.40.2.3 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.

19.41 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)

19.41.1 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.

19.41.2 Function Documentation

19.41.2.1 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.

19.41.2.2 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.

19.41.2.3 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.

19.42 Interoperability with Intel Xeon Phi (MIC)

Functions

- static int hwloc_intel_mic_get_device_cpuset (hwloc_topology_t topology, int idx, hwloc_cpuset_t set)
- static hwloc_obj_t hwloc_intel_mic_get_device_osdev_by_index (hwloc_topology_t topology, unsigned idx)

19.42.1 Detailed Description

This interface offers ways to retrieve topology information about Intel Xeon Phi (MIC) devices.

19.42.2 Function Documentation

19.42.2.1 static int hwloc_intel_mic_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 MIC device whose index is `idx`.

Return the CPU set describing the locality of the MIC device whose index is `idx`.

Topology `topology` and device index `idx` 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_intel_mic_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.

19.42.2.2 static hwloc_obj_t hwloc_intel_mic_get_device_osdev_by_index (hwloc_topology_t *topology,* unsigned *idx*) [inline], [static]

Get the hwloc OS device object corresponding to the MIC device for the given index.

Return the OS device object describing the MIC 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 must be enabled in the topology.

Note

The corresponding PCI device object can be obtained by looking at the OS device parent object.

19.43 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)

19.43.1 Detailed Description

This interface offers ways to retrieve topology information about OpenFabrics devices (InfiniBand, Omni-Path, usNIC, etc).

19.43.2 Function Documentation

19.43.2.1 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.

19.43.2.2 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.

19.43.2.3 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.

19.44 Interoperability with Myrinet Express

Functions

- static int hwloc_mx_board_get_device_cpuset (hwloc_topology_t topology, unsigned id, hwloc_cpuset_t set)
- static int hwloc_mx_endpoint_get_device_cpuset (hwloc_topology_t topology, mx_endpoint_t endpoint, hwloc_↩ cpuset_t set)

19.44.1 Detailed Description

This interface offers ways to retrieve topology information about Myrinet Express hardware.

19.44.2 Function Documentation

19.44.2.1 static int hwloc_mx_board_get_device_cpuset (hwloc_topology_t *topology,* unsigned *id,* hwloc_cpuset_t *set*) [inline], [static]

Get the CPU set of logical processors that are physically close the MX board `id`.

Return the CPU set describing the locality of the Myrinet Express board whose index is `id`.

Topology `topology` and device `id` must match the local machine. I/O devices detection is not needed in the topology.

The function only returns the locality of the device. No additional information about the device is available.

19.44.2.2 static int hwloc_mx_endpoint_get_device_cpuset (hwloc_topology_t *topology,* mx_endpoint_t *endpoint,* hwloc_cpuset_t *set*) [inline], [static]

Get the CPU set of logical processors that are physically close the MX endpoint `endpoint`.

Return the CPU set describing the locality of the Myrinet Express board that runs the MX endpoint `endpoint`.

Topology `topology` and device `id` must match the local machine. I/O devices detection is not needed in the topology.

The function only returns the locality of the endpoint. No additional information about the endpoint or device is available.

Chapter 20

Data Structure Documentation

20.1 hwloc_backend Struct Reference

`#include <plugins.h>`

Data Fields

- unsigned long flags
- int is_custom
- int is_thissystem
- void * private_data
- void(* disable)(struct hwloc_backend *backend)
- int(* discover)(struct hwloc_backend *backend)
- int(* get_obj_cpuset)(struct hwloc_backend *backend, struct hwloc_backend *caller, struct hwloc_obj *obj, hwloc_bitmap_t cpuset)
- int(* notify_new_object)(struct hwloc_backend *backend, struct hwloc_backend *caller, struct hwloc_obj *obj)

20.1.1 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().

20.1.2 Field Documentation

20.1.2.1 void(* hwloc_backend::disable)(struct hwloc_backend *backend)

Callback for freeing the private_data. May be NULL.

20.1.2.2 int(* hwloc_backend::discover)(struct hwloc_backend *backend)

Main discovery callback. returns > 0 if it modified the topology tree, -1 on error, 0 otherwise. May be NULL if type is HWLOC_DISC_COMPONENT_TYPE_MISC.

20.1.2.3 unsigned long hwloc_backend::flags

Backend flags, as an OR'ed set of hwloc_backend_flag_e.

20.1.2.4 int(* hwloc_backend::get_obj_cpuset)(struct hwloc_backend *backend, struct hwloc_backend *caller, struct hwloc_obj *obj, hwloc_bitmap_t cpuset)

Callback used by the PCI backend to retrieve the locality of a PCI object from the OS/cpu backend. May be NULL.

20.1.2.5 int hwloc_backend::is_custom

Backend-specific 'is_custom' property. Shortcut on !strcmp(..->component->name, "custom"). Only the custom component should touch this.

20.1.2.6 int hwloc_backend::is_thissystem

Backend-specific 'is_thissystem' property. Set to 0 or 1 if the backend should enforce the thissystem flag when it gets enabled. Set to -1 if the backend doesn't care (default).

20.1.2.7 int(* hwloc_backend::notify_new_object)(struct hwloc_backend *backend, struct hwloc_backend *caller, struct hwloc_obj *obj)

Callback called by backends to notify this backend that a new object was added. returns > 0 if it modified the topology tree, 0 otherwise. May be NULL.

20.1.2.8 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

20.2 hwloc_obj_attr_u::hwloc_bridge_attr_s Struct Reference

`#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

20.2.1 Detailed Description

Bridge specific Object Attribues.

20.2.2 Field Documentation

20.2.2.1 unsigned hwloc_obj_attr_u::hwloc_bridge_attr_s::depth

20.2.2.2 unsigned short hwloc_obj_attr_u::hwloc_bridge_attr_s::domain

20.2.2.3 union { ... } hwloc_obj_attr_u::hwloc_bridge_attr_s::downstream

20.2.2.4 hwloc_obj_bridge_type_t hwloc_obj_attr_u::hwloc_bridge_attr_s::downstream_type

20.2.2.5 struct { ... } hwloc_obj_attr_u::hwloc_bridge_attr_s::pci

20.2.2.6 struct hwloc_pcidev_attr_s hwloc_obj_attr_u::hwloc_bridge_attr_s::pci

20.2.2.7 unsigned char hwloc_obj_attr_u::hwloc_bridge_attr_s::secondary_bus

20.2.2.8 unsigned char hwloc_obj_attr_u::hwloc_bridge_attr_s::subordinate_bus

20.2.2.9 union { ... } hwloc_obj_attr_u::hwloc_bridge_attr_s::upstream

20.2.2.10 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

20.3 hwloc_obj_attr_u::hwloc_cache_attr_s Struct Reference

```
#include <hwloc.h>
```

Data Fields

- hwloc_uint64_t size

- unsigned depth
- unsigned linesize
- int associativity
- hwloc_obj_cache_type_t type

20.3.1 Detailed Description

Cache-specific Object Attributes.

20.3.2 Field Documentation

20.3.2.1 int hwloc_obj_attr_u::hwloc_cache_attr_s::associativity

Ways of associativity, -1 if fully associative, 0 if unknown.

20.3.2.2 unsigned hwloc_obj_attr_u::hwloc_cache_attr_s::depth

Depth of cache (e.g., L1, L2, ...etc.)

20.3.2.3 unsigned hwloc_obj_attr_u::hwloc_cache_attr_s::linesize

Cache-line size in bytes. 0 if unknown.

20.3.2.4 hwloc_uint64_t hwloc_obj_attr_u::hwloc_cache_attr_s::size

Size of cache in bytes.

20.3.2.5 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

20.4 hwloc_component Struct Reference

```
#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

20.4.1 Detailed Description

Generic component structure.

Generic components structure, either statically listed by configure in static-components.h or dynamically loaded as a plugin.

20.4.2 Field Documentation

20.4.2.1 unsigned hwloc_component::abi

Component ABI version, set to HWLOC_COMPONENT_ABI.

20.4.2.2 void* hwloc_component::data

Component data, pointing to a struct hwloc_disc_component or struct hwloc_xml_component.

20.4.2.3 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.

20.4.2.4 unsigned long hwloc_component::flags

Component flags, unused for now.

20.4.2.5 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.

20.4.2.6 hwloc_component_type_t hwloc_component::type

Component type.

The documentation for this struct was generated from the following file:

- plugins.h

20.5 hwloc_disc_component Struct Reference

`#include <plugins.h>`

Data Fields

- hwloc_disc_component_type_t type
- const char * name
- unsigned excludes
- struct hwloc_backend *(* instantiate)(struct hwloc_disc_component *component, const void *data1, const void *data2, const void *data3)
- unsigned priority

20.5.1 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.

20.5.2 Field Documentation

20.5.2.1 unsigned hwloc_disc_component::excludes

Component types to exclude, as an OR'ed set of hwloc_disc_component_type_e.

For a GLOBAL component, this usually includes all other types (~0).

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.

20.5.2.2 struct hwloc_backend*(* hwloc_disc_component::instantiate)(struct hwloc_disc_component *component, const void *data1, const void *data2, const void *data3)

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().

20.5.2.3 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.

20.5.2.4 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/custom), 20 for pci, 10 for other misc components (opencl etc.).

20.5.2.5 hwloc_disc_component_type_t hwloc_disc_component::type

Discovery component type.

The documentation for this struct was generated from the following file:

- plugins.h

20.6 hwloc_distances_s Struct Reference

`#include <hwloc.h>`

Data Fields

- unsigned relative_depth
- unsigned nbobjs
- float * latency
- float latency_max
- float latency_base

20.6.1 Detailed Description

Distances between objects.

One object may contain a distance structure describing distances between all its descendants at a given relative depth. If the containing object is the root object of the topology, then the distances are available for all objects in the machine.

If the `latency` pointer is not `NULL`, the pointed array contains memory latencies (non-zero values), see below.

In the future, some other types of distances may be considered. In these cases, `latency` may be `NULL`.

20.6.2 Field Documentation

20.6.2.1 float* hwloc_distances_s::latency

Matrix of latencies between objects, stored as a one-dimension array. May be `NULL` if the distances considered here are not latencies.

Unless defined by the user, this currently 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 accurate. It corresponds to the latency for accessing the memory of one node from a core in another node.

Values are normalized to get 1.0 as the minimal value in the matrix. Latency from i-th to j-th object is stored in slot i*nbobjs+j.

20.6.2.2 float hwloc_distances_s::latency_base

The multiplier that should be applied to latency matrix to retrieve the original OS-provided latencies. Usually 10 on Linux since ACPI SLIT uses 10 for local latency.

20.6.2.3 float hwloc_distances_s::latency_max

The maximal value in the latency matrix.

20.6.2.4 unsigned hwloc_distances_s::nbobjs

Number of objects considered in the matrix. It is the number of descendant objects at `relative_depth` below the containing object. It corresponds to the result of hwloc_get_nbobjs_inside_cpuset_by_depth().

20.6.2.5 unsigned hwloc_distances_s::relative_depth

Relative depth of the considered objects below the object containing this distance information.

The documentation for this struct was generated from the following file:

- hwloc.h

20.7 hwloc_obj_attr_u::hwloc_group_attr_s Struct Reference

`#include <hwloc.h>`

Data Fields

- unsigned depth

20.7.1 Detailed Description

Group-specific Object Attributes.

20.7.2 Field Documentation

20.7.2.1 unsigned hwloc_obj_attr_u::hwloc_group_attr_s::depth

Depth of group object.

The documentation for this struct was generated from the following file:

- hwloc.h

20.8 hwloc_obj Struct Reference

`#include <hwloc.h>`

Data Fields

- hwloc_obj_type_t type
- unsigned os_index
- char * name
- struct hwloc_obj_memory_s memory
- union hwloc_obj_attr_u * attr
- unsigned depth
- unsigned logical_index
- signed os_level
- 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
- unsigned arity
- struct hwloc_obj ** children
- struct hwloc_obj * first_child
- struct hwloc_obj * last_child
- void * userdata
- hwloc_cpuset_t cpuset
- hwloc_cpuset_t complete_cpuset
- hwloc_cpuset_t online_cpuset
- hwloc_cpuset_t allowed_cpuset
- hwloc_nodeset_t nodeset
- hwloc_nodeset_t complete_nodeset
- hwloc_nodeset_t allowed_nodeset
- struct hwloc_distances_s ** distances
- unsigned distances_count
- struct hwloc_obj_info_s * infos
- unsigned infos_count
- int symmetric_subtree

20.8.1 Detailed Description

Structure of a topology object.

Applications must not modify any field except hwloc_obj.userdata.

20.8.2 Field Documentation

20.8.2.1 hwloc_cpuset_t hwloc_obj::allowed_cpuset

The CPU set of allowed logical processors.

This includes the CPUs contained in this object which are allowed for binding, i.e. passing them to the hwloc binding functions should not return permission errors. This is usually restricted by administration rules. Some of them may however be offline so binding to them may still not be possible, see online_cpuset.

Note

Its value must not be changed, hwloc_bitmap_dup() must be used instead.

20.8.2.2 hwloc_nodeset_t hwloc_obj::allowed_nodeset

The set of allowed NUMA memory nodes.

This includes the NUMA memory nodes contained in this object which are allowed for memory allocation, i.e. passing them to NUMA node-directed memory allocation should not return permission errors. This is usually restricted by administration rules.

If there are no NUMA nodes in the machine, all the memory is close to this object, so `allowed_nodeset` is full.

Note

Its value must not be changed, hwloc_bitmap_dup() must be used instead.

20.8.2.3 unsigned hwloc_obj::arity

Number of children.

20.8.2.4 union hwloc_obj_attr_u* hwloc_obj::attr

Object type-specific Attributes, may be `NULL` if no attribute value was found.

20.8.2.5 struct hwloc_obj** hwloc_obj::children

Children, `children`[0 .. arity -1].

20.8.2.6 hwloc_cpuset_t hwloc_obj::complete_cpuset

The complete CPU set of logical processors of this object,.

This includes not only the same as the cpuset field, but also the CPUs for which topology information is unknown or incomplete, and the CPUs that are ignored when the HWLOC_TOPOLOGY_FLAG_WHOLE_SYSTEM 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.

20.8.2.7 hwloc_nodeset_t hwloc_obj::complete_nodeset

The complete NUMA node set of this object,.

This includes not only the same as the nodeset field, but also the NUMA nodes for which topology information is unknown or incomplete, and the nodes that are ignored when the HWLOC_TOPOLOGY_FLAG_WHOLE_SYSTEM flag is not set. Thus no corresponding 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 `complete_nodeset` is full.

Note

Its value must not be changed, hwloc_bitmap_dup() must be used instead.

20.8.2.8 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_WHOLE_SYSTEM configuration flag is set, some of these CPUs may be offline, or not allowed for binding, see online_cpuset and allowed_cpuset.

Note

Its value must not be changed, hwloc_bitmap_dup() must be used instead.

20.8.2.9 unsigned hwloc_obj::depth

Vertical index in the hierarchy. 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.

20.8.2.10 struct hwloc_distances_s** hwloc_obj::distances

Distances between all objects at same depth below this object.

20.8.2.11 unsigned hwloc_obj::distances_count

20.8.2.12 struct hwloc_obj* hwloc_obj::first_child

First child.

20.8.2.13 struct hwloc_obj_info_s* hwloc_obj::infos

Array of stringified info type=name.

20.8.2.14 unsigned hwloc_obj::infos_count

Size of infos array.

20.8.2.15 struct hwloc_obj* hwloc_obj::last_child

Last child.

20.8.2.16 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.

20.8.2.17 struct hwloc_obj_memory_s hwloc_obj::memory

Memory attributes.

20.8.2.18 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.

20.8.2.19 struct hwloc_obj* hwloc_obj::next_cousin

Next object of same type and depth.

20.8.2.20 struct hwloc_obj* hwloc_obj::next_sibling

Next object below the same parent.

20.8.2.21 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 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 NODE objects).

In the end, these nodes are those that are close to the current object.

If the HWLOC_TOPOLOGY_FLAG_WHOLE_SYSTEM configuration flag is set, some of these nodes may not be allowed for allocation, see allowed_nodeset.

If there are no NUMA nodes in the machine, all the memory is close to this object, so `nodeset` is full.

Note

Its value must not be changed, hwloc_bitmap_dup() must be used instead.

20.8.2.22 hwloc_cpuset_t hwloc_obj::online_cpuset

The CPU set of online logical processors.

This includes the CPUs contained in this object that are online, i.e. draw power and can execute threads. It may however not be allowed to bind to them due to administration rules, see allowed_cpuset.

Note

Its value must not be changed, hwloc_bitmap_dup() must be used instead.

20.8.2.23 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.

20.8.2.24 signed hwloc_obj::os_level

OS-provided physical level, -1 if unknown or meaningless.

20.8.2.25 struct hwloc_obj* hwloc_obj::parent

Parent, NULL if root (system object)

20.8.2.26 struct hwloc_obj* hwloc_obj::prev_cousin

Previous object of same type and depth.

20.8.2.27 struct hwloc_obj* hwloc_obj::prev_sibling

Previous object below the same parent.

20.8.2.28 unsigned hwloc_obj::sibling_rank

Index in parent's children[] array.

20.8.2.29 int hwloc_obj::symmetric_subtree

Set if the subtree of objects below this object is symmetric, which means all children and their children have identical subtrees. If set in the topology root object, lstopo may export the topology as a synthetic string.

20.8.2.30 hwloc_obj_type_t hwloc_obj::type

Type of object.

20.8.2.31 void* hwloc_obj::userdata

Application-given private data pointer, initialized to NULL, use it as you wish. See hwloc_topology_set_userdata_↩ export_callback() if you wish to export this field to XML.

The documentation for this struct was generated from the following file:

- hwloc.h

20.9 hwloc_obj_attr_u Union Reference

`#include <hwloc.h>`

Data Structures

- struct hwloc_bridge_attr_s
- struct hwloc_cache_attr_s
- struct hwloc_group_attr_s
- struct hwloc_osdev_attr_s
- struct hwloc_pcidev_attr_s

Data Fields

- 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

20.9.1 Detailed Description

Object type-specific Attributes.

20.9.2 Field Documentation

20.9.2.1 struct hwloc_obj_attr_u::hwloc_bridge_attr_s hwloc_obj_attr_u::bridge

20.9.2.2 struct hwloc_obj_attr_u::hwloc_cache_attr_s hwloc_obj_attr_u::cache

20.9.2.3 struct hwloc_obj_attr_u::hwloc_group_attr_s hwloc_obj_attr_u::group

20.9.2.4 struct hwloc_obj_attr_u::hwloc_osdev_attr_s hwloc_obj_attr_u::osdev

20.9.2.5 struct hwloc_obj_attr_u::hwloc_pcidev_attr_s hwloc_obj_attr_u::pcidev

The documentation for this union was generated from the following file:

- hwloc.h

20.10 hwloc_obj_info_s Struct Reference

```
#include <hwloc.h>
```

Data Fields

- char * name
- char * value

20.10.1 Detailed Description

Object info.

See also

Consulting and Adding Key-Value Info Attributes

20.10.2 Field Documentation

20.10.2.1 char* hwloc_obj_info_s::name

Info name.

20.10.2.2 char* hwloc_obj_info_s::value

Info value.

The documentation for this struct was generated from the following file:

- hwloc.h

20.11 hwloc_obj_memory_s::hwloc_obj_memory_page_type_s Struct Reference

`#include <hwloc.h>`

Data Fields

- hwloc_uint64_t size
- hwloc_uint64_t count

20.11.1 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.

20.11.2 Field Documentation

20.11.2.1 hwloc_uint64_t hwloc_obj_memory_s::hwloc_obj_memory_page_type_s::count

Number of pages of this size.

20.11.2.2 hwloc_uint64_t hwloc_obj_memory_s::hwloc_obj_memory_page_type_s::size

Size of pages.

The documentation for this struct was generated from the following file:

- hwloc.h

20.12 hwloc_obj_memory_s Struct Reference

`#include <hwloc.h>`

Data Structures

- struct hwloc_obj_memory_page_type_s

Data Fields

- hwloc_uint64_t total_memory
- hwloc_uint64_t local_memory
- unsigned page_types_len
- struct hwloc_obj_memory_s::hwloc_obj_memory_page_type_s * page_types

20.12.1 Detailed Description

Object memory.

20.12.2 Field Documentation

20.12.2.1 hwloc_uint64_t hwloc_obj_memory_s::local_memory

Local memory (in bytes)

20.12.2.2 struct hwloc_obj_memory_s::hwloc_obj_memory_page_type_s * hwloc_obj_memory_s::page_types

20.12.2.3 unsigned hwloc_obj_memory_s::page_types_len

Size of array `page_types`.

20.12.2.4 hwloc_uint64_t hwloc_obj_memory_s::total_memory

Total memory (in bytes) in this object and its children.

The documentation for this struct was generated from the following file:

- hwloc.h

20.13 hwloc_obj_attr_u::hwloc_osdev_attr_s Struct Reference

`#include <hwloc.h>`

Data Fields

- hwloc_obj_osdev_type_t type

20.13.1 Detailed Description

OS Device specific Object Attributes.

20.13.2 Field Documentation

20.13.2.1 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

20.14 hwloc_obj_attr_u::hwloc_pcidev_attr_s Struct Reference

`#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

20.14.1 Detailed Description

PCI Device specific Object Attributes.

20.14.2 Field Documentation

20.14.2.1 unsigned char hwloc_obj_attr_u::hwloc_pcidev_attr_s::bus

20.14.2.2 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::class_id

20.14.2.3 unsigned char hwloc_obj_attr_u::hwloc_pcidev_attr_s::dev

20.14.2.4 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::device_id

20.14.2.5 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::domain

20.14.2.6 unsigned char hwloc_obj_attr_u::hwloc_pcidev_attr_s::func

20.14.2.7 float hwloc_obj_attr_u::hwloc_pcidev_attr_s::linkspeed

20.14.2.8 unsigned char hwloc_obj_attr_u::hwloc_pcidev_attr_s::revision

20.14.2.9 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::subdevice_id

20.14.2.10 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::subvendor_id

20.14.2.11 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

20.15 hwloc_topology_cpubind_support Struct Reference

`#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

20.15.1 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).

20.15.2 Field Documentation

20.15.2.1 unsigned char hwloc_topology_cpubind_support::get_proc_cpubind

Getting the binding of a whole given process is supported.

20.15.2.2 unsigned char hwloc_topology_cpubind_support::get_proc_last_cpu_location

Getting the last processors where a whole process ran is supported

20.15.2.3 unsigned char hwloc_topology_cpubind_support::get_thisproc_cpubind

Getting the binding of the whole current process is supported.

20.15.2.4 unsigned char hwloc_topology_cpubind_support::get_thisproc_last_cpu_location

Getting the last processors where the whole current process ran is supported

20.15.2.5 unsigned char hwloc_topology_cpubind_support::get_thisthread_cpubind

Getting the binding of the current thread only is supported.

20.15.2.6 unsigned char hwloc_topology_cpubind_support::get_thisthread_last_cpu_location

Getting the last processors where the current thread ran is supported

20.15.2.7 unsigned char hwloc_topology_cpubind_support::get_thread_cpubind

Getting the binding of a given thread only is supported.

20.15.2.8 unsigned char hwloc_topology_cpubind_support::set_proc_cpubind

Binding a whole given process is supported.

20.15.2.9 unsigned char hwloc_topology_cpubind_support::set_thisproc_cpubind

Binding the whole current process is supported.

20.15.2.10 unsigned char hwloc_topology_cpubind_support::set_thisthread_cpubind

Binding the current thread only is supported.

20.15.2.11 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

20.16 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

20.16.1 Field Documentation

20.16.1.1 union hwloc_topology_diff_u* hwloc_topology_diff_u::hwloc_topology_diff_generic_s::next

20.16.1.2 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

20.17 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

20.17.1 Field Documentation

20.17.1.1 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

20.18 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
- unsigned obj_depth
- unsigned obj_index
- union hwloc_topology_diff_obj_attr_u diff

20.18.1 Field Documentation

20.18.1.1 union hwloc_topology_diff_obj_attr_u hwloc_topology_diff_u::hwloc_topology_diff_obj_attr_s::diff

20.18.1.2 union hwloc_topology_diff_u* hwloc_topology_diff_u::hwloc_topology_diff_obj_attr_s::next

20.18.1.3 unsigned hwloc_topology_diff_u::hwloc_topology_diff_obj_attr_s::obj_depth

20.18.1.4 unsigned hwloc_topology_diff_u::hwloc_topology_diff_obj_attr_s::obj_index

20.18.1.5 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

20.19 hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_string_s Struct Reference

`#include <diff.h>`

Data Fields

- hwloc_topology_diff_obj_attr_type_t type
- char * name
- char * oldvalue
- char * newvalue

20.19.1 Detailed Description

String attribute modification with an optional name.

20.19.2 Field Documentation

20.19.2.1 char* hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_string_s::name

20.19.2.2 char* hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_string_s::newvalue

20.19.2.3 char* hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_string_s::oldvalue

20.19.2.4 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

20.20 hwloc_topology_diff_obj_attr_u Union Reference

`#include <diff.h>`

Data Structures

- struct hwloc_topology_diff_obj_attr_generic_s
- struct hwloc_topology_diff_obj_attr_string_s
- struct hwloc_topology_diff_obj_attr_uint64_s

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

20.20.1 Detailed Description

One object attribute difference.

20.20.2 Field Documentation

20.20.2.1 struct hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_generic_s hwloc_topology_diff_obj_attr_u::generic

20.20.2.2 struct hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_string_s hwloc_topology_diff_obj_attr_u::string

20.20.2.3 struct hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_uint64_s hwloc_topology_diff_obj_attr_u::uint64

The documentation for this union was generated from the following file:

- diff.h

20.21 hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_uint64_s Struct Reference

```
#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

20.21.1 Detailed Description

Integer attribute modification with an optional index.

20.21.2 Field Documentation

20.21.2.1 hwloc_uint64_t hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_uint64_s::index

20.21.2.2 hwloc_uint64_t hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_uint64_s::newvalue

20.21.2.3 hwloc_uint64_t hwloc_topology_diff_obj_attr_u::hwloc_topology_diff_obj_attr_uint64_s::oldvalue

20.21.2.4 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

20.22 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
- unsigned obj_depth
- unsigned obj_index

20.22.1 Field Documentation

20.22.1.1 union hwloc_topology_diff_u* hwloc_topology_diff_u::hwloc_topology_diff_too_complex_s::next

20.22.1.2 unsigned hwloc_topology_diff_u::hwloc_topology_diff_too_complex_s::obj_depth

20.22.1.3 unsigned hwloc_topology_diff_u::hwloc_topology_diff_too_complex_s::obj_index

20.22.1.4 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

20.23 hwloc_topology_diff_u Union Reference

`#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

20.23.1 Detailed Description

One element of a difference list between two topologies.

20.23.2 Field Documentation

20.23.2.1 struct hwloc_topology_diff_u::hwloc_topology_diff_generic_s hwloc_topology_diff_u::generic

20.23.2.2 struct hwloc_topology_diff_u::hwloc_topology_diff_obj_attr_s hwloc_topology_diff_u::obj_attr

20.23.2.3 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

20.24 hwloc_topology_discovery_support Struct Reference

```
#include <hwloc.h>
```

Data Fields

- unsigned char pu

20.24.1 Detailed Description

Flags describing actual discovery support for this topology.

20.24.2 Field Documentation

20.24.2.1 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

20.25 hwloc_topology_membind_support Struct Reference

`#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 replicate_membind
- unsigned char nexttouch_membind
- unsigned char migrate_membind
- unsigned char get_area_memlocation

20.25.1 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).

20.25.2 Field Documentation

20.25.2.1 unsigned char hwloc_topology_membind_support::alloc_membind

Allocating a bound memory area is supported.

20.25.2.2 unsigned char hwloc_topology_membind_support::bind_membind

Bind policy is supported.

20.25.2.3 unsigned char hwloc_topology_membind_support::firsttouch_membind

First-touch policy is supported.

20.25.2.4 unsigned char hwloc_topology_membind_support::get_area_membind

Getting the binding of a given memory area is supported.

20.25.2.5 unsigned char hwloc_topology_membind_support::get_area_memlocation

Getting the last NUMA nodes where a memory area was allocated is supported

20.25.2.6 unsigned char hwloc_topology_membind_support::get_proc_membind

Getting the binding of a whole given process is supported.

20.25.2.7 unsigned char hwloc_topology_membind_support::get_thisproc_membind

Getting the binding of the whole current process is supported.

20.25.2.8 unsigned char hwloc_topology_membind_support::get_thisthread_membind

Getting the binding of the current thread only is supported.

20.25.2.9 unsigned char hwloc_topology_membind_support::interleave_membind

Interleave policy is supported.

20.25.2.10 unsigned char hwloc_topology_membind_support::migrate_membind

Migration flags is supported.

20.25.2.11 unsigned char hwloc_topology_membind_support::nexttouch_membind

Next-touch migration policy is supported.

20.25.2.12 unsigned char hwloc_topology_membind_support::replicate_membind

Replication policy is supported.

20.25.2.13 unsigned char hwloc_topology_membind_support::set_area_membind

Binding a given memory area is supported.

20.25.2.14 unsigned char hwloc_topology_membind_support::set_proc_membind

Binding a whole given process is supported.

20.25.2.15 unsigned char hwloc_topology_membind_support::set_thisproc_membind

Binding the whole current process is supported.

20.25.2.16 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

20.26 hwloc_topology_support Struct Reference

`#include <hwloc.h>`

Data Fields

- struct hwloc_topology_discovery_support * discovery
- struct hwloc_topology_cpubind_support * cpubind
- struct hwloc_topology_membind_support * membind

20.26.1 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.

20.26.2 Field Documentation

20.26.2.1 struct hwloc_topology_cpubind_support* hwloc_topology_support::cpubind

20.26.2.2 struct hwloc_topology_discovery_support* hwloc_topology_support::discovery

20.26.2.3 struct hwloc_topology_membind_support* hwloc_topology_support::membind

The documentation for this struct was generated from the following file:

- hwloc.h

Index