Hardware Locality (hwloc)

1.4.3

Generated by Doxygen 1.8.3.1

Sun Feb 24 2013 16:05:15

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 sockets, 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)
- Solaris
- AIX
- Darwin / OS X
- FreeBSD and its variants, such as kFreeBSD/GNU
- OSF/1 (a.k.a., Tru64)
- HP-UX
- Microsoft Windows

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 could be used for other OSes too).

To check whether hwloc works on a particular machine, just try to build it and run `lstopo`. 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 checked out of Subversion:

```
shell$ svn checkout http://svn.open-mpi.org/svn/hwloc/trunk hwloc-trunk
shell$ cd hwloc-trunk
shell$ ./autogen.sh
```

Note that GNU Autoconf >=2.63, Automake >=1.10 and Libtool >=2.2.6 are required when building from a Subversion checkout.

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 can be found 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.
- 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.
- hwloc can use one of two different libraries for I/O device discovery:
    1. libpciaccess (BSD).
    2. libpci, from the pciutils package (GPL).

Note that because of the possibility of GPL taint (remember that hwloc is BSD-licensed), hwloc's `configure` script will prefer `libpciaccess` to the `pciutils` package. Indeed, if `libpciaccess` is not found, hwloc will not use `pciutils` unless it is specifically requested via the `--enable-libpci` flag is provided.

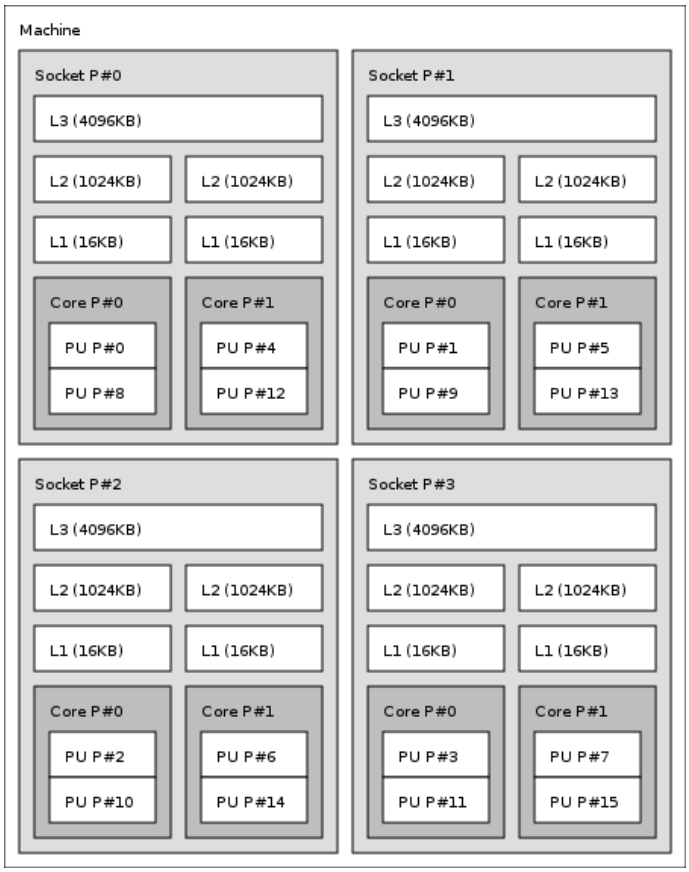
Also note that if you install supplemental libraries in non-standard locations, hwloc's configure script may not be able to find them without some help. You may need to specify additional CPPFLAGS, LDFLAGS, or PKG_CONFIG_PATH values on the configure command line.

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-socket 2-core machine with hyperthreading, the `lstopo` tool may show the following graphical output:



Here's the equivalent output in textual form:

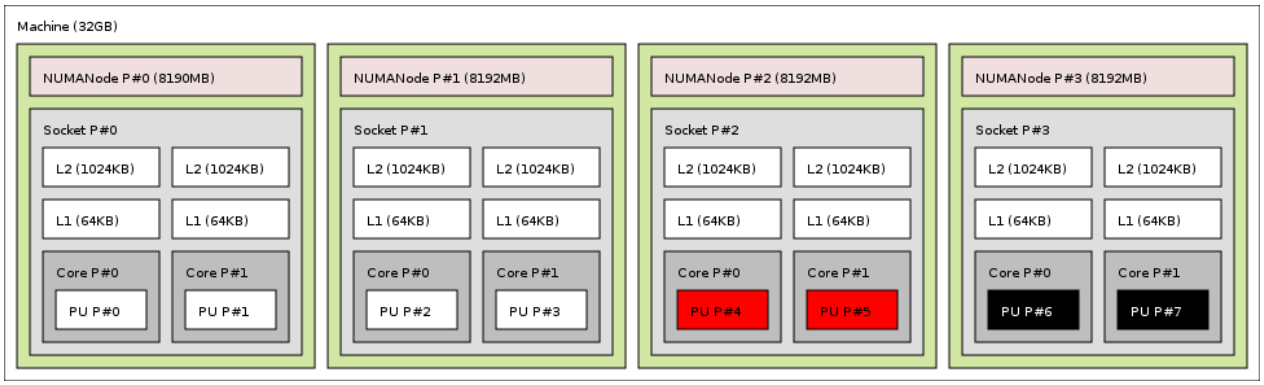
```
Machine (16GB)
  Socket L#0 + L3 L#0 (4096KB)
    L2 L#0 (1024KB) + L1 L#0 (16KB) + Core L#0
      PU L#0 (P#0)
      PU L#1 (P#8)
    L2 L#1 (1024KB) + L1 L#1 (16KB) + Core L#1
      PU L#2 (P#4)
      PU L#3 (P#12)
  Socket L#1 + L3 L#1 (4096KB)
    L2 L#2 (1024KB) + L1 L#2 (16KB) + Core L#2
      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)
  Socket 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)
  Socket 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)
```

Finally, here's the equivalent output in XML. Long lines were artificially broken for document clarity (in the real output, each XML tag is on a single line), and only socket #0 is shown for brevity:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topology SYSTEM "hwloc.dtd">
<topology>
  <object type="Machine" os_level="-1" os_index="0" cpuset="0x0000ffff"
      complete_cpuset="0x0000ffff" online_cpuset="0x0000ffff"
      allowed_cpuset="0x0000ffff"
      dmi_board_vendor="Dell Computer Corporation" dmi_board_name="0RD318"
      local_memory="16648183808">
    <page_type size="4096" count="4064498"/>
    <page_type size="2097152" count="0"/>
    <object type="Socket" os_level="-1" os_index="0" cpuset="0x00001111"
        complete_cpuset="0x00001111" online_cpuset="0x00001111"
        allowed_cpuset="0x00001111">
      <object type="Cache" os_level="-1" cpuset="0x00001111"
          complete_cpuset="0x00001111" online_cpuset="0x00001111"
          allowed_cpuset="0x00001111" cache_size="4194304" depth="3"
          cache_linesize="64">
        <object type="Cache" os_level="-1" cpuset="0x00000101"
            complete_cpuset="0x00000101" online_cpuset="0x00000101"
            allowed_cpuset="0x00000101" cache_size="1048576" depth="2"
            cache_linesize="64">
          <object type="Cache" os_level="-1" cpuset="0x00000101"
              complete_cpuset="0x00000101" online_cpuset="0x00000101"
              allowed_cpuset="0x00000101" cache_size="16384" depth="1"
              cache_linesize="64">
            <object type="Core" os_level="-1" os_index="0" cpuset="0x00000101"
                complete_cpuset="0x00000101" online_cpuset="0x00000101"
                allowed_cpuset="0x00000101">
              <object type="PU" os_level="-1" os_index="0" cpuset="0x00000001"
                  complete_cpuset="0x00000001" online_cpuset="0x00000001"
                  allowed_cpuset="0x00000001"/>
```

```
            <object type="PU" os_level="-1" os_index="8" cpuset="0x00000100"
                complete_cpuset="0x00000100" online_cpuset="0x00000100"
                allowed_cpuset="0x00000100"/>
            </object>
          </object>
        </object>
        <object type="Cache" os_level="-1" cpuset="0x00001010"
            complete_cpuset="0x00001010" online_cpuset="0x00001010"
            allowed_cpuset="0x00001010" cache_size="1048576" depth="2"
            cache_linesize="64">
          <object type="Cache" os_level="-1" cpuset="0x00001010"
              complete_cpuset="0x00001010" online_cpuset="0x00001010"
              allowed_cpuset="0x00001010" cache_size="16384" depth="1"
              cache_linesize="64">
            <object type="Core" os_level="-1" os_index="1" cpuset="0x00001010"
                complete_cpuset="0x00001010" online_cpuset="0x00001010"
                allowed_cpuset="0x00001010">
              <object type="PU" os_level="-1" os_index="4" cpuset="0x00000010"
                  complete_cpuset="0x00000010" online_cpuset="0x00000010"
                  allowed_cpuset="0x00000010"/>
              <object type="PU" os_level="-1" os_index="12" cpuset="0x00001000"
                  complete_cpuset="0x00001000" online_cpuset="0x00001000"
                  allowed_cpuset="0x00001000"/>
            </object>
          </object>
        </object>
      </object>
    </object>
    <!-- ...other sockets listed here ... -->
  </object>
</topology>
```

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



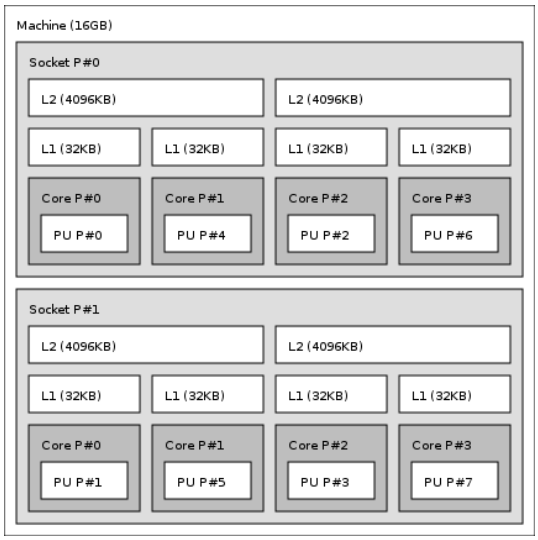
Here's the equivalent output in textual form:

```
Machine (32GB)
  NUMANode L#0 (P#0 8190MB) + Socket 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) + Socket 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) + Socket 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) + Socket 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)
```

And here's the equivalent output in XML. Similar to above, line breaks were added and only PU #0 is shown for brevity:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topology SYSTEM "hwloc.dtd">
<topology>
  <object type="Machine" os_level="-1" os_index="0" cpuset="0x000000ff"
      complete_cpuset="0x000000ff" online_cpuset="0x000000ff"
      allowed_cpuset="0x000000ff" nodeset="0x000000ff"
      complete_nodeset="0x000000ff" allowed_nodeset="0x000000ff"
      dmi_board_vendor="TYAN Computer Corp" dmi_board_name="S4881 ">
    <page_type size="4096" count="0"/>
    <page_type size="2097152" count="0"/>
    <object type="NUMANode" os_level="-1" os_index="0" cpuset="0x00000003"
        complete_cpuset="0x00000003" online_cpuset="0x00000003"
        allowed_cpuset="0x00000003" nodeset="0x00000001"
        complete_nodeset="0x00000001" allowed_nodeset="0x00000001"
        local_memory="7514177536">
      <page_type size="4096" count="1834516"/>
      <page_type size="2097152" count="0"/>
      <object type="Socket" os_level="-1" os_index="0" cpuset="0x00000003"
          complete_cpuset="0x00000003" online_cpuset="0x00000003"
          allowed_cpuset="0x00000003" nodeset="0x00000001"
          complete_nodeset="0x00000001" allowed_nodeset="0x00000001">
        <object type="Cache" os_level="-1" cpuset="0x00000001"
            complete_cpuset="0x00000001" online_cpuset="0x00000001"
            allowed_cpuset="0x00000001" nodeset="0x00000001"
            complete_nodeset="0x00000001" allowed_nodeset="0x00000001"
            cache_size="1048576" depth="2" cache_linesize="64">
          <object type="Cache" os_level="-1" cpuset="0x00000001"
              complete_cpuset="0x00000001" online_cpuset="0x00000001"
              allowed_cpuset="0x00000001" nodeset="0x00000001"
              complete_nodeset="0x00000001" allowed_nodeset="0x00000001"
              cache_size="65536" depth="1" cache_linesize="64">
            <object type="Core" os_level="-1" os_index="0"
                cpuset="0x00000001" complete_cpuset="0x00000001"
                online_cpuset="0x00000001" allowed_cpuset="0x00000001"
                nodeset="0x00000001" complete_nodeset="0x00000001"
                allowed_nodeset="0x00000001">
              <object type="PU" os_level="-1" os_index="0" cpuset="0x00000001"
                  complete_cpuset="0x00000001" online_cpuset="0x00000001"
                  allowed_cpuset="0x00000001" nodeset="0x00000001"
                  complete_nodeset="0x00000001" allowed_nodeset="0x00000001"/>
            </object>
          </object>
        </object>
  <!-- ...more objects listed here ... -->
</topology>
```

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



Here's the same output in textual form:

```
Machine (16GB)
  Socket 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)
  Socket 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)
```

And the same output in XML (line breaks added, only PU #0 shown):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topology SYSTEM "hwloc.dtd">
<topology>
  <object type="Machine" os_level="-1" os_index="0" cpuset="0x000000ff"
      complete_cpuset="0x000000ff" online_cpuset="0x000000ff"
      allowed_cpuset="0x000000ff" dmi_board_vendor="Dell Inc."
      dmi_board_name="0NR282" local_memory="16865292288">
    <page_type size="4096" count="4117503"/>
    <page_type size="2097152" count="0"/>
    <object type="Socket" os_level="-1" os_index="0" cpuset="0x00000055"
        complete_cpuset="0x00000055" online_cpuset="0x00000055"
        allowed_cpuset="0x00000055">
      <object type="Cache" os_level="-1" cpuset="0x00000011"
          complete_cpuset="0x00000011" online_cpuset="0x00000011"
          allowed_cpuset="0x00000011" cache_size="4194304" depth="2"
          cache_linesize="64">
        <object type="Cache" os_level="-1" cpuset="0x00000001"
            complete_cpuset="0x00000001" online_cpuset="0x00000001"
            allowed_cpuset="0x00000001" cache_size="32768" depth="1"
            cache_linesize="64">
          <object type="Core" os_level="-1" os_index="0" cpuset="0x00000001"
              complete_cpuset="0x00000001" online_cpuset="0x00000001"
              allowed_cpuset="0x00000001">
            <object type="PU" os_level="-1" os_index="0" cpuset="0x00000001"
                complete_cpuset="0x00000001" online_cpuset="0x00000001"
```

```
              allowed_cpuset="0x00000001"/>
          </object>
        </object>
        <object type="Cache" os_level="-1" cpuset="0x00000010"
            complete_cpuset="0x00000010" online_cpuset="0x00000010"
            allowed_cpuset="0x00000010" cache_size="32768" depth="1"
            cache_linesize="64">
          <object type="Core" os_level="-1" os_index="1" cpuset="0x00000010"
              complete_cpuset="0x00000010" online_cpuset="0x00000010"
              allowed_cpuset="0x00000010">
            <object type="PU" os_level="-1" os_index="4" cpuset="0x00000010"
                complete_cpuset="0x00000010" online_cpuset="0x00000010"
                allowed_cpuset="0x00000010"/>
          </object>
        </object>
      </object>
  <!-- ...more objects listed here ... -->
</topology>
```

1.4 Programming Interface

The basic interface is available in hwloc.h. It essentially offers low-level routines for advanced programmers that want to manually manipulate objects and follow links between them. Documentation for everything in hwloc.h are provided later in this document. Developers should also look at hwloc/helper.h (and also in 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/.

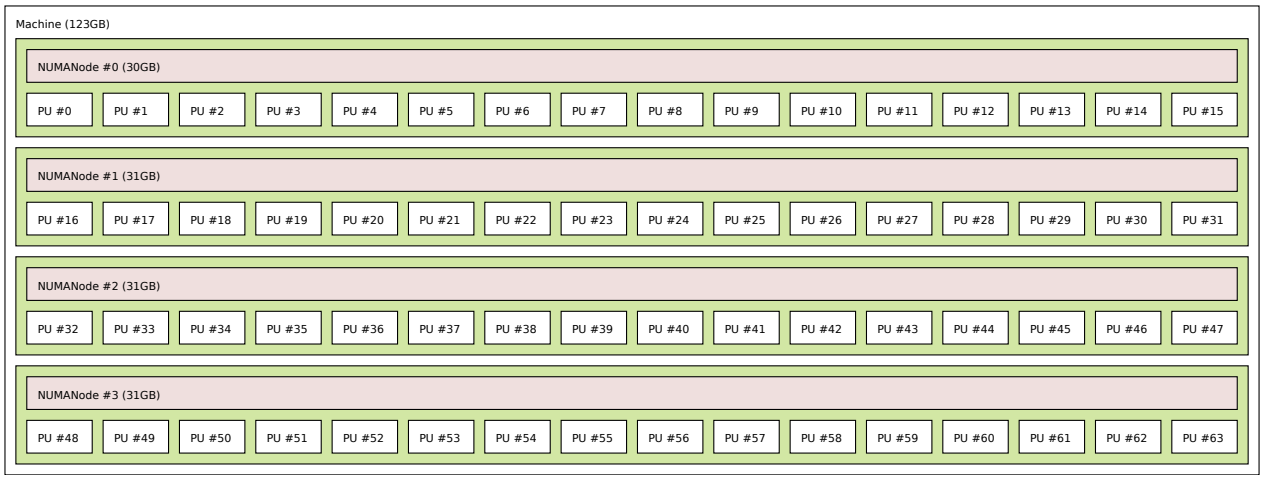
NOTE: If you are building the documentation from a Subversion checkout, you will need to have Doxygen and pdflatex installed – the documentation will be built during the normal "make" process. The documentation is installed during "make install" to $prefix/share/doc/hwloc/ and your systems default man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

1.4.1 Portability

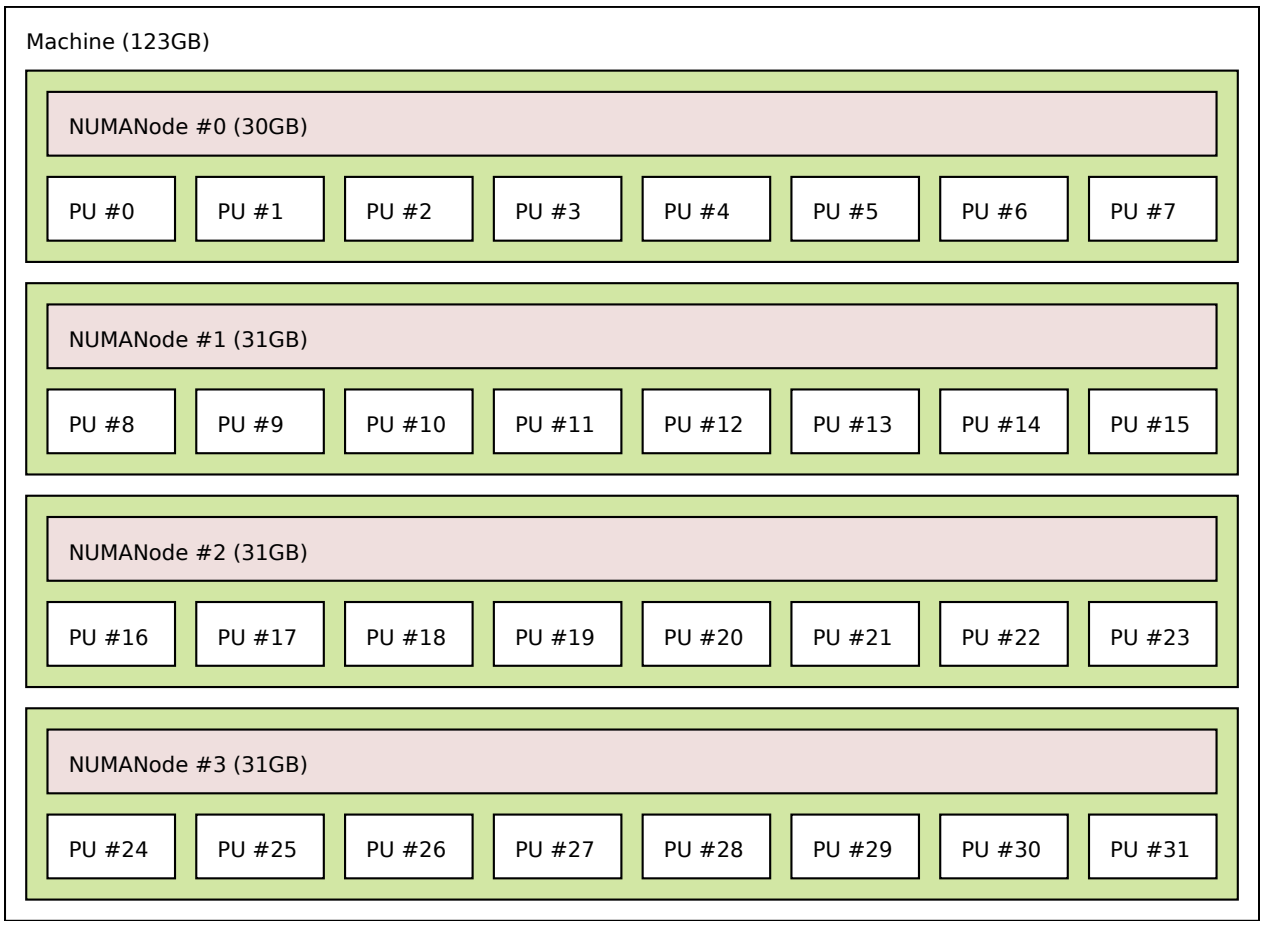
As shown in CLI Examples, hwloc can obtain information on a wide variety of hardware topologies. However, some platforms and/or operating system versions will only report a subset of this information. For example, on an PPC64-based system with 32 cores (each with 2 hardware threads) running a default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

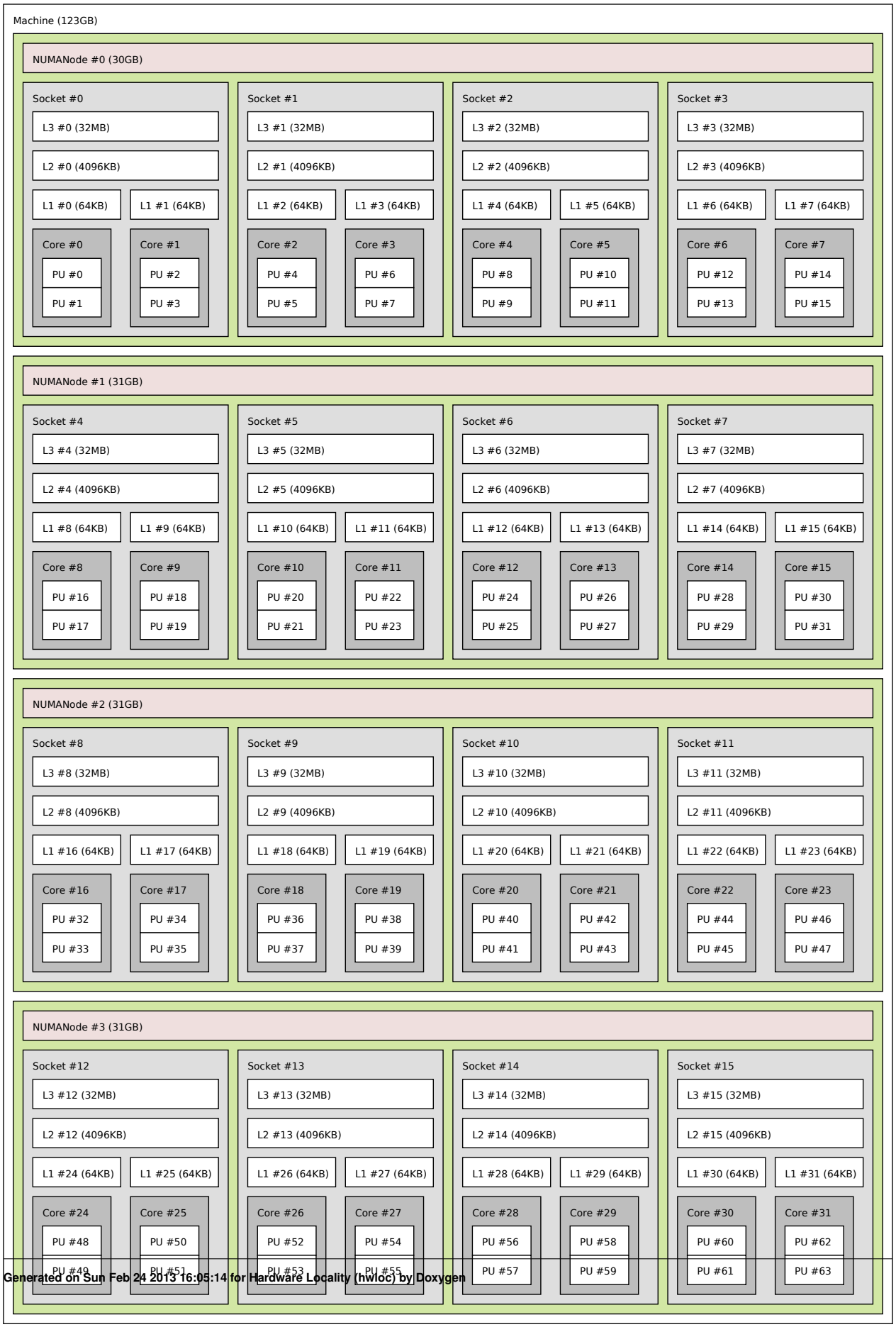


Notice that hwloc only sees half the PUs when SMT is disabled. PU #15, for example, seems to change location from NUMA node #0 to #1. In reality, no PUs "moved" – they were simply re-numbered when hwloc only saw half as many. Hence, PU #15 in the SMT-disabled picture probably corresponds to PU #30 in 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 (123GB)
NUMANode #0 (30GB)
Socket #0
L3 #0 (32MB)
L2 #0 (4096KB)
L1 #0 (64KB)
L1 #1 (64KB)
Core #0
PU #0
PU #1
Core #1
PU #2
PU #3
Socket #1
L3 #1 (32MB)
L2 #1 (4096KB)
L1 #2 (64KB)
L1 #3 (64KB)
Core #2
PU #4
PU #5
Core #3
PU #6
PU #7
Socket #2
L3 #2 (32MB)
L2 #2 (4096KB)
L1 #4 (64KB)
L1 #5 (64KB)
Core #4
PU #8
PU #9
Core #5
PU #10
PU #11
Socket #3
L3 #3 (32MB)
L2 #3 (4096KB)
L1 #6 (64KB)
L1 #7 (64KB)
Core #6
PU #12
PU #13
Core #7
PU #14
PU #15
NUMANode #1 (31GB)
Socket #4
L3 #4 (32MB)
L2 #4 (4096KB)
L1 #8 (64KB)
L1 #9 (64KB)
Core #8
PU #16
PU #17
Core #9
PU #18
PU #19
Socket #5
L3 #5 (32MB)
L2 #5 (4096KB)
L1 #10 (64KB)
L1 #11 (64KB)
Core #10
PU #20
PU #21
Core #11
PU #22
PU #23
Socket #6
L3 #6 (32MB)
L2 #6 (4096KB)
L1 #12 (64KB)
L1 #13 (64KB)
Core #12
PU #24
PU #25
Core #13
PU #26
PU #27
Socket #7
L3 #7 (32MB)
L2 #7 (4096KB)
L1 #14 (64KB)
L1 #15 (64KB)
Core #14
PU #28
PU #29
Core #15
PU #30
PU #31
NUMANode #2 (31GB)
Socket #8
L3 #8 (32MB)
L2 #8 (4096KB)
L1 #16 (64KB)
L1 #17 (64KB)
Core #16
PU #32
PU #33
Core #17
PU #34
PU #35
Socket #9
L3 #9 (32MB)
L2 #9 (4096KB)
L1 #18 (64KB)
L1 #19 (64KB)
Core #18
PU #36
PU #37
Core #19
PU #38
PU #39
Socket #10
L3 #10 (32MB)
L2 #10 (4096KB)
L1 #20 (64KB)
L1 #21 (64KB)
Core #20
PU #40
PU #41
Core #21
PU #42
PU #43
Socket #11
L3 #11 (32MB)
L2 #11 (4096KB)
L1 #22 (64KB)
L1 #23 (64KB)
Core #22
PU #44
PU #45
Core #23
PU #46
PU #47
NUMANode #3 (31GB)
Socket #12
L3 #12 (32MB)
L2 #12 (4096KB)
L1 #24 (64KB)
L1 #25 (64KB)
Core #24
PU #48
PU #49
Core #25
PU #50
PU #51
Socket #13
L3 #13 (32MB)
L2 #13 (4096KB)
L1 #26 (64KB)
L1 #27 (64KB)
Core #26
PU #52
PU #53
Core #27
PU #54
PU #55
Socket #14
L3 #14 (32MB)
L2 #14 (4096KB)
L1 #28 (64KB)
L1 #29 (64KB)
Core #28
PU #56
PU #57
Core #29
PU #58
PU #59
Socket #15
L3 #15 (32MB)
L2 #15 (4096KB)
L1 #30 (64KB)
L1 #31 (64KB)
Core #30
PU #60
PU #61
Core #31
PU #62
PU #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.

```
/* Example hwloc API program.
 *
 * Copyright © 2009-2010 inria.  All rights reserved.
 * Copyright © 2009-2011 Université Bordeaux 1
 * 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 string[128];
    unsigned i;

    hwloc_obj_snprintf(string, sizeof(string), topology, obj, "#", 0);
    printf("%*s%s\n", 2*depth, "", string);
    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_snprintf(string, sizeof(string), topology,
                    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 sockets.
 *****************************************************************/
depth = hwloc_get_type_depth(topology, HWLOC_OBJ_SOCKET);
if (depth == HWLOC_TYPE_DEPTH_UNKNOWN) {
    printf("*** The number of sockets is unknown\n");
} else {
    printf("*** %u socket(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/hyperthreaded). */
    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_NODE);
    if (n) {
        void *m;
        size = 1024*1024;

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

        m = malloc(size);
        hwloc_set_area_membind_nodeset(topology, m, size, obj->
      nodeset,
                  HWLOC_MEMBIND_DEFAULT, 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 sockets – each socket 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: Socket#0
Index 1: Socket#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)
  Socket#0
    Core#0
      PU#0
    Core#1
      PU#1
```

```
  Socket#1
    Core#3
      PU#2
    Core#2
      PU#3
*** 2 socket(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://svn.open-mpi.org/trac/hwloc/).

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.sh` 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
- Multi-node Topologies
- Importing and exporting topologies from/to XML files

- Interoperability With Other Software
- Thread Safety
- 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 caches) are missing in only some parts of the machine.

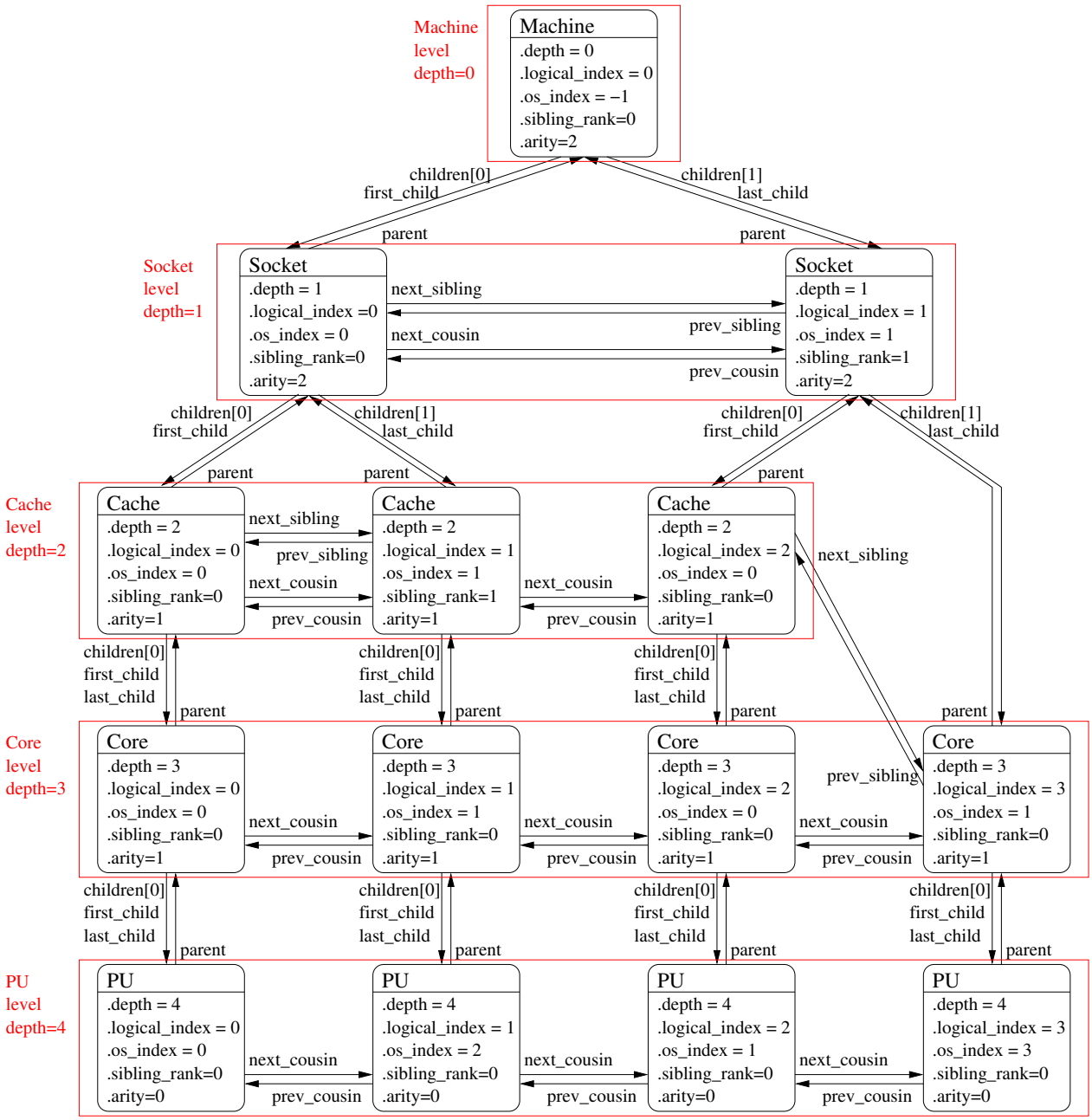
OS or physical index The index that the operating system (OS) uses to identify the object. This may be completely arbitrary, non-unique, non-contiguous, not representative of logical proximity, and may depend on the BIOS configuration. That is why hwloc almost never uses them, only in the default lstopo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

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).

Logical processor

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 SMT processor. "Logical processor" should not be confused with "Logical index of a processor". "Logical processor" is only one of the names which can be found in various documentations to designate a processing unit.

The following diagram can help to understand the vocabulary of the relationships by showing the example of a machine with two dual core sockets (with no hardware threads); thus, a topology with 4 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.



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

See also What happens if my topology is asymmetric? for more details.

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

lstopo (also known as hwloc-info and hwloc-ls) displays the hierarchical topology map of the current system. The output may be graphical or textual, and can also be exported to numerous file formats such as PDF, PNG, XML, and others.

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 sockets 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.

3.3 hwloc-calc

hwloc-calc is generally used to create bitmap strings to pass to hwloc-bind. Although hwloc-bind accepts many forms of object specification (i.e., bitmap strings are one of many forms that hwloc-bind understands), they can be useful, compact representations in shell scripts, for example.

hwloc-calc generates bitmap strings from given hardware objects with the ability to aggregate them, intersect them, and more. hwloc-calc generally uses the same syntax than hwloc-bind, but multiple instances may be composed to generate complex combinations.

Note that hwloc-calc can also generate lists of logical processors or NUMA nodes that are convenient to pass to some external tools such as taskset or numactl.

3.4 hwloc-distrib

hwloc-distrib generates a set of bitmap strings 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.

3.5 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.

3.6 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.

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-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.

3.9 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.

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 enable 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.

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.

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`, `Socket`, `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.

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.

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.

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.

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 pciutils library is needed to detect PCI devices and bridges, and the actual locality of these devices is only currently detected on Linux. Also, some operating systems require privileges for probing PCI devices, see Does hwloc require privileged access? for details.

6.2 I/O object hierarchy

When I/O discovery is enabled and supported, some additional objects (types HWLOC_OBJ_BRIDGE, HWLOC_OBJ_PCI_DEVICE and HWLOC_OBJ_OS_DEVICE) are added to the topology as a child of the object they are close to. For instance, if a I/O Hub is connected to a socket, the corresponding hwloc bridge object (and its PCI bridges and devices children) is inserted as a child of the corresponding hwloc socket object.

These new objects have neither CPU sets nor node sets (NULL pointers) because they are not directly usable by the user applications. 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 Advanced I/O object traversal helpers).

An I/O hierarchy is organized as follows: A hostbridge object (HWLOC_OBJ_BRIDGE object with upstream type *Host* and downstream type *PCI*) is attached below a regular object (usually the entire machine or a NUMA node). There may be multiple hostbridges in the machine, attached to different places, but all I/O 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.

6.3 Software devices

Although each PCI device is uniquely identified by its bus ID (e.g. 0000:01:02.3), the application can hardly find out which PCI device is actually used when manipulating software handle (such as the *eth0* network interface or the *mlx4_0* OpenFabrics HCA). Therefore hwloc tries to add software devices (`HWLOC_OBJ_OS_DEVICE`) below their PCI objects. These objects can be identified by their usual operating system-wide names, e.g. *eth0* or *mlx4_0*. However, this ability is currently only available on Linux for some classes of devices. It should especially be noted that proprietary graphics driver currently do not create any interesting software device for GPUs, they should therefore be manipulated as PCI device objects. On the contrary some PCI devices may contain multiple software device (see the example below).

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

6.4 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 Advanced I/O object traversal helpers).

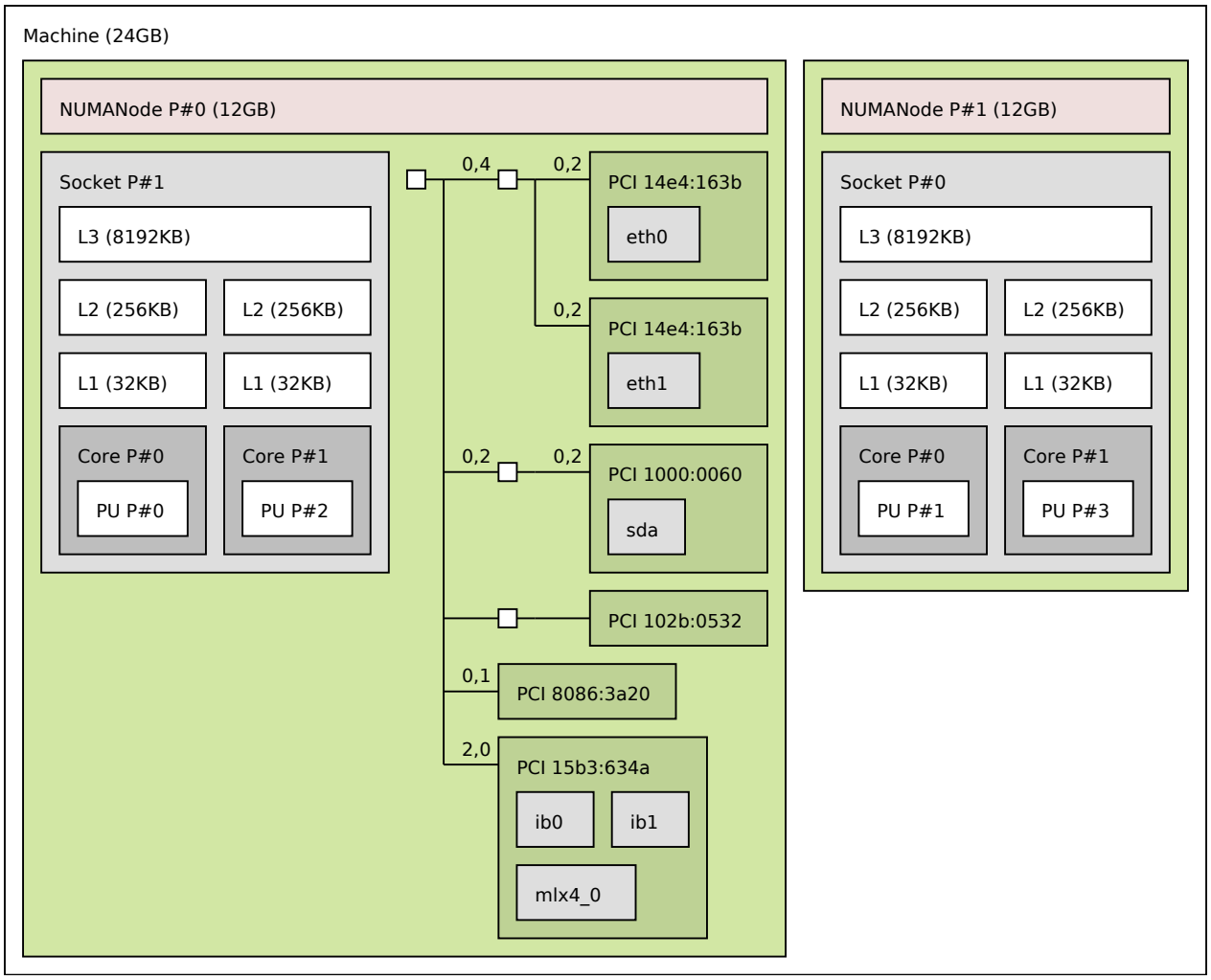
I/O objects do not actually contain any locality information because their CPU sets and node sets are NULL. Their locality must be retrieved by walking up the object tree (through the `parent` link) until an non-I/O object is found (see hwloc_get_non_io_ancestor_obj()). This regular object should have 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. For instance, `pci=0000:02:03.0` (respectively `os=eth0`) is replaced by the set of CPUs that are close to this PCI device (respectively software device). This enables easy binding of I/O-intensive applications near the device they use.

6.5 Examples

The following picture shows a dual-socket dual-core host whose PCI bus is connected to the first socket 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 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)
    Socket 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) + Socket 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

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.

7.1 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 correspond too, 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.

7.2 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).

7.3 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.

7.4 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 8

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 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.

8.1 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.

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.

8.2 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 9

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 nodemasks and 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_setaffinity().

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. Note that if I/O device discovery is enabled, such devices may also appear as PCI objects and as OS objects in the topology.

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.

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. Note that if I/O device discovery is enabled, GPUs may also appear as PCI objects in the topology.

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 10

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 Create and Destroy Topologies) 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 Tinker With Topologies.) 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 Configure Topology Detection) do not modify the topology directly, but they do modify internal structures describing the behavior of the next invocation of `hwloc_topology_load()`. Hence, all of these functions should not be used concurrently.

Note that these functions do not modify the current topology until it is actually reloaded; it is possible to use them while other threads are only read the current topology.

Chapter 11

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).

11.1 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.

11.2 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 12

Frequently Asked Questions

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

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

For this purpose, hwloc offers XML import/export 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_THISSYSTEM environment variable should also be set to 1 to assert that loaded file is really the underlying system.

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

12.2 Does hwloc require privileged access?

hwloc discovers the topology by querying the operating system. Some minor features may require privileged access to the operation system. For instance PCI link speed discovery on Linux is reserved to root, and the entire PCI discovery on FreeBSD requires access to the /dev/pci special file.

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.

12.3 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.

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

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#1
    PU L#2 (P#1)
    PU L#3 (P#3)
```

If you need to ignore symmetric multithreading, 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 socket */
hwloc_obj_t socket, core;
socket = hwloc_get_obj_by_type(topology, HWLOC_OBJ_SOCKET, 0);
core = hwloc_get_obj_inside_cpuset_by_type(topology, socket->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
```

12.5 What happens if my topology is asymmetric?

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

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 usually annotated with a depth or level attribute (for instance L2 cache). In this case, this attribute is also taken in account when gathering objects as horizontal levels. To be clear: there will be one level for L1 caches, another level for L2 caches, etc.

If the topology is asymmetric (e.g., if a cache is missing in one of the 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.

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 levels).**

12.6 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 kept intact as long as the object is valid, which means as long as topology objects are not modified by reloading or restricting the topology.

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

It is also possible to insert Misc objects with custom names anywhere in the topology (`hwloc_topology_insert_misc_object_by_cpuset()`) or as a leaf of the topology (`hwloc_topology_insert_misc_object_by_parent()`).

12.7 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
```

12.8 How do I handle 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.

To check whether hwloc is at least 1.2, you should use:

```
#include <hwloc.h>
#if HWLOC_API_VERSION >= 0x00010200
...
#endif
```

One of the major changes in hwloc 1.1 was the addition of the bitmap API. It supersedes the now deprecated cpuset API which will be removed in a future hwloc release. It is strongly recommended to switch existing codes to the bitmap API. Keeping support for older hwloc versions is easy. For instance, if your code uses `hwloc_cpuset_alloc`, you should use `hwloc_bitmap_alloc` instead and add the following code to one of your common headers:

```
#include <hwloc.h>
#if HWLOC_API_VERSION < 0x00010100
#define hwloc_bitmap_alloc hwloc_cpuset_alloc
#endif
```

Similarly, the hwloc 1.0 interface may be detected by comparing `HWLOC_API_VERSION` with `0x00010000`.

hwloc 0.9 did not define any `HWLOC_API_VERSION` but this very old release probably does not deserve support from your application anymore.

Chapter 13

Module Index

13.1 Modules

Here is a list of all modules:

Chapter 14

Data Structure Index

14.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 15

Module Documentation

15.1 API version

Macros

- #define HWLOC_API_VERSION 0x00010400

Functions

- HWLOC_DECLSPEC unsigned hwloc_get_api_version (void)

15.1.1 Detailed Description

15.1.2 Macro Definition Documentation

15.1.2.1 #define HWLOC_API_VERSION 0x00010400

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

15.1.3 Function Documentation

15.1.3.1 HWLOC_DECLSPEC unsigned hwloc_get_api_version (void)

Indicate at runtime which hwloc API version was used at build time.

15.2 Topology context

Typedefs

- typedef struct hwloc_topology * hwloc_topology_t

15.2.1 Detailed Description

15.2.2 Typedef Documentation

15.2.2.1 typedef struct hwloc_topology* hwloc_topology_t

Topology context.

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

15.3 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

15.3.1 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.

15.3.2 Typedef Documentation

15.3.2.1 typedef hwloc_const_bitmap_t hwloc_const_cpuset_t

A non-modifiable hwloc_cpuset_t.

15.3.2.2 typedef hwloc_const_bitmap_t hwloc_const_nodeset_t

A non-modifiable hwloc_nodeset_t.

15.3.2.3 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).

15.3.2.4 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).

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 Conversion between cpuset and nodeset.

15.4 Topology Object Types

Typedefs

- 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_NODE, HWLOC_OBJ_SOCKET, 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_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 }
- enum hwloc_compare_types_e { HWLOC_TYPE_UNORDERED }

Functions

- HWLOC_DECLSPEC int hwloc_compare_types (hwloc_obj_type_t type1, hwloc_obj_type_t type2) __hwloc_attribute_const

15.4.1 Detailed Description

15.4.2 Typedef Documentation

15.4.2.1 typedef enum hwloc_obj_bridge_type_e hwloc_obj_bridge_type_t

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

15.4.2.2 typedef enum hwloc_obj_osdev_type_e hwloc_obj_osdev_type_t

Type of a OS device.

15.4.3 Enumeration Type Documentation

15.4.3.1 enum hwloc_compare_types_e

Enumerator

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

15.4.3.2 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.

15.4.3.3 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 the "card0" DRM device on Linux.

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

HWLOC_OBJ_OSDEV_OPENFABRICS Operating system openfabrics device. For instance the "mlx4_0" InfiniBand HCA device on Linux.

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

15.4.3.4 enum hwloc_obj_type_t

Type of topology object.

Note

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

Enumerator

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

HWLOC_OBJ_SOCKET Socket, physical package, or chip. In the physical meaning, i.e. that you can add or remove physically.

HWLOC_OBJ_CACHE Data cache. Can be L1, 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. 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.

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

15.4.4 Function Documentation

15.4.4.1 HWLOC_DECLSPEC int hwloc_compare_types (hwloc_obj_type_t *type1,* hwloc_obj_type_t *type2*) const

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 sockets 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 sockets may also contain nodes. This is thus just to be seen as a fallback comparison method.

15.5 Topology Objects

Data Structures

- struct hwloc_obj_memory_s
- struct hwloc_obj
- union hwloc_obj_attr_u
- struct hwloc_distances_s
- struct hwloc_obj_info_s

Typedefs

- typedef struct hwloc_obj * hwloc_obj_t

15.5.1 Detailed Description

15.5.2 Typedef Documentation

15.5.2.1 typedef struct hwloc_obj* hwloc_obj_t

Convenience typedef; a pointer to a struct hwloc_obj.

15.6 Create and Destroy Topologies

Functions

- HWLOC_DECLSPEC int hwloc_topology_init (hwloc_topology_t *topologyp)
- HWLOC_DECLSPEC int hwloc_topology_load (hwloc_topology_t topology)
- HWLOC_DECLSPEC void hwloc_topology_destroy (hwloc_topology_t topology)
- HWLOC_DECLSPEC void hwloc_topology_check (hwloc_topology_t topology)

15.6.1 Detailed Description

15.6.2 Function Documentation

15.6.2.1 HWLOC_DECLSPEC 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().

15.6.2.2 HWLOC_DECLSPEC void hwloc_topology_destroy (hwloc_topology_t *topology*)

Terminate and free a topology context.

Parameters

topology	is the topology to be freed

15.6.2.3 HWLOC_DECLSPEC 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.

15.6.2.4 HWLOC_DECLSPEC 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 Configure Topology Detection 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.

See Also

Configure Topology Detection

15.7 Configure Topology Detection

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 = (1<<2), HWLOC_TOPOLOGY_FLAG_IO_BRIDGES = (1<<3), HWLOC_TOPOLOGY_FLAG_WHOLE_IO = (1<<4) }

Functions

- HWLOC_DECLSPEC int hwloc_topology_ignore_type (hwloc_topology_t topology, hwloc_obj_type_t type)
- HWLOC_DECLSPEC int hwloc_topology_ignore_type_keep_structure (hwloc_topology_t topology, hwloc_obj_type_t type)
- HWLOC_DECLSPEC int hwloc_topology_ignore_all_keep_structure (hwloc_topology_t topology)
- HWLOC_DECLSPEC int hwloc_topology_set_flags (hwloc_topology_t topology, unsigned long flags)
- HWLOC_DECLSPEC int hwloc_topology_set_pid (hwloc_topology_t __hwloc_restrict topology, hwloc_pid_t pid)
- HWLOC_DECLSPEC int hwloc_topology_set_fsroot (hwloc_topology_t __hwloc_restrict topology, const char *__hwloc_restrict fsroot_path)
- HWLOC_DECLSPEC int hwloc_topology_set_synthetic (hwloc_topology_t __hwloc_restrict topology, const char *__hwloc_restrict description)
- HWLOC_DECLSPEC int hwloc_topology_set_xml (hwloc_topology_t __hwloc_restrict topology, const char *__hwloc_restrict xmlpath)
- HWLOC_DECLSPEC int hwloc_topology_set_xmlbuffer (hwloc_topology_t __hwloc_restrict topology, const char *__hwloc_restrict buffer, int size)
- HWLOC_DECLSPEC int hwloc_topology_set_custom (hwloc_topology_t topology)
- HWLOC_DECLSPEC int hwloc_topology_set_distance_matrix (hwloc_topology_t __hwloc_restrict topology, hwloc_obj_type_t type, unsigned nbobjs, unsigned *os_index, float *distances)
- HWLOC_DECLSPEC struct hwloc_topology_support * hwloc_topology_get_support (hwloc_topology_t __hwloc_restrict topology)

15.7.1 Detailed Description

These 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().

15.7.2 Enumeration Type Documentation

15.7.2.1 enum hwloc_topology_flags_e

Flags to be set onto a topology context before load.

Flags should be given to hwloc_topology_set_flags().

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 Cpusets and gather all processors and memory nodes, and ignore the fact that some resources may be offline.

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 libpci 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. 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) and bridges (even those that have no device behind them) using the libpci backend.

15.7.3 Function Documentation

15.7.3.1 HWLOC_DECLSPEC struct hwloc_topology_support* hwloc_topology_get_support (hwloc_topology_t __hwloc_restrict *topology*) [read]

Retrieve the topology support.

15.7.3.2 HWLOC_DECLSPEC 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: Each ignored object should have a single children or be the only child of its parent. I/O objects may not be ignored, topology flags should be used to configure their discovery instead.

15.7.3.3 HWLOC_DECLSPEC 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. I/O objects may not be ignored, topology flags should be used to configure their discovery instead.

15.7.3.4 HWLOC_DECLSPEC 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.

15.7.3.5 HWLOC_DECLSPEC int hwloc_topology_set_custom (hwloc_topology_t *topology*)

Prepare the topology for custom assembly.

The topology then contains a single root object. It may 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.

15.7.3.6 HWLOC_DECLSPEC int hwloc_topology_set_distance_matrix (hwloc_topology_t __hwloc_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. 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.

15.7.3.7 HWLOC_DECLSPEC 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.

15.7.3.8 HWLOC_DECLSPEC int hwloc_topology_set_fsroot (hwloc_topology_t __hwloc_restrict *topology*, const char *__hwloc_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.
The existing topology is cleared even on failure.

15.7.3.9 HWLOC_DECLSPEC int hwloc_topology_set_pid (hwloc_topology_t __hwloc_restrict *topology,* hwloc_pid_t *pid*)

Change which pid 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.

15.7.3.10 HWLOC_DECLSPEC int hwloc_topology_set_synthetic (hwloc_topology_t __hwloc_restrict *topology,* const char *__hwloc_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.

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.
The existing topology is cleared even on failure.

15.7.3.11 HWLOC_DECLSPEC int hwloc_topology_set_xml (hwloc_topology_t __hwloc_restrict *topology,* const char *__hwloc_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

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.
The existing topology is cleared even on failure.

15.7.3.12 HWLOC_DECLSPEC int hwloc_topology_set_xmlbuffer (hwloc_topology_t __hwloc_restrict *topology*, const char *__hwloc_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

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.
The existing topology is cleared even on failure.

15.8 Tinker With Topologies.

Enumerations

- enum hwloc_restrict_flags_e { HWLOC_RESTRICT_FLAG_ADAPT_DISTANCES, HWLOC_RESTRICT_FLAG_ADAPT_MISC, HWLOC_RESTRICT_FLAG_ADAPT_IO }

Functions

- HWLOC_DECLSPEC int hwloc_topology_export_xml (hwloc_topology_t topology, const char *xmlpath)
- HWLOC_DECLSPEC int hwloc_topology_export_xmlbuffer (hwloc_topology_t topology, char **xmlbuffer, int *buflen)
- HWLOC_DECLSPEC void hwloc_free_xmlbuffer (hwloc_topology_t topology, char *xmlbuffer)
- HWLOC_DECLSPEC hwloc_obj_t hwloc_topology_insert_misc_object_by_cpuset (hwloc_topology_t topology, hwloc_const_cpuset_t cpuset, const char *name)
- HWLOC_DECLSPEC hwloc_obj_t hwloc_topology_insert_misc_object_by_parent (hwloc_topology_t topology, hwloc_obj_t parent, const char *name)
- HWLOC_DECLSPEC int hwloc_topology_restrict (hwloc_topology_t __hwloc_restrict topology, hwloc_const_cpuset_t cpuset, unsigned long flags)

15.8.1 Detailed Description

15.8.2 Enumeration Type Documentation

15.8.2.1 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.

15.8.3 Function Documentation

15.8.3.1 HWLOC_DECLSPEC void hwloc_free_xmlbuffer (hwloc_topology_t *topology*, char * *xmlbuffer*)

Free a buffer allocated by hwloc_topology_export_xmlbuffer()

15.8.3.2 HWLOC_DECLSPEC 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.

15.8.3.3 HWLOC_DECLSPEC 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.

15.8.3.4 HWLOC_DECLSPEC 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.

15.8.3.5 HWLOC_DECLSPEC 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

15.8.3.6 HWLOC_DECLSPEC int hwloc_topology_restrict (hwloc_topology_t __hwloc_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 reloading the entire topology with hwloc_topology_load().

15.9 Get Some Topology Information

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

- HWLOC_DECLSPEC unsigned hwloc_topology_get_depth (hwloc_topology_t __hwloc_restrict topology) __hwloc_attribute_pure
- HWLOC_DECLSPEC int hwloc_get_type_depth (hwloc_topology_t topology, hwloc_obj_type_t type)
- HWLOC_DECLSPEC hwloc_obj_type_t hwloc_get_depth_type (hwloc_topology_t topology, unsigned depth) __hwloc_attribute_pure
- HWLOC_DECLSPEC unsigned hwloc_get_nbobjs_by_depth (hwloc_topology_t topology, unsigned depth) __hwloc_attribute_pure
- static __hwloc_inline int hwloc_get_nbobjs_by_type (hwloc_topology_t topology, hwloc_obj_type_t type) __hwloc_attribute_pure
- HWLOC_DECLSPEC int hwloc_topology_is_thissystem (hwloc_topology_t __hwloc_restrict topology) __hwloc_attribute_pure

15.9.1 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 socket has fewer caches than its peers.

15.9.2 Enumeration Type Documentation

15.9.2.1 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.

15.9.3 Function Documentation

15.9.3.1 HWLOC_DECLSPEC hwloc_obj_type_t hwloc_get_depth_type (hwloc_topology_t *topology,* unsigned *depth*)

Returns the type of objects at depth `depth`.

Returns

-1 if depth `depth` does not exist.

15.9.3.2 HWLOC_DECLSPEC unsigned hwloc_get_nbobjs_by_depth (hwloc_topology_t *topology,* unsigned *depth*)

Returns the width of level at depth `depth`.

15.9.3.3 static __hwloc_inline int hwloc_get_nbobjs_by_type (hwloc_topology_t *topology,* hwloc_obj_type_t *type*) [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.

15.9.3.4 HWLOC_DECLSPEC 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, the function returns HWLOC_-TYPE_DEPTH_MULTIPLE.

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.

15.9.3.5 HWLOC_DECLSPEC unsigned hwloc_topology_get_depth (hwloc_topology_t __hwloc_restrict *topology*)

Get the depth of the hierarchical tree of objects.

This is the depth of HWLOC_OBJ_PU objects plus one.

15.9.3.6 HWLOC_DECLSPEC int hwloc_topology_is_thissystem (hwloc_topology_t __hwloc_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).

15.10 Retrieve Objects

Functions

- HWLOC_DECLSPEC hwloc_obj_t hwloc_get_obj_by_depth (hwloc_topology_t topology, unsigned depth, unsigned idx) __hwloc_attribute_pure
- static __hwloc_inline hwloc_obj_t hwloc_get_obj_by_type (hwloc_topology_t topology, hwloc_obj_type_t type, unsigned idx) __hwloc_attribute_pure

15.10.1 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 socket has fewer caches than its peers.

15.10.2 Function Documentation

15.10.2.1 HWLOC_DECLSPEC 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`.

15.10.2.2 static __hwloc_inline hwloc_obj_t hwloc_get_obj_by_type (hwloc_topology_t *topology,* hwloc_obj_type_t *type,* unsigned *idx*) [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().

15.11 Object/String Conversion

Functions

- HWLOC_DECLSPEC const char * hwloc_obj_type_string (hwloc_obj_type_t type) __hwloc_attribute_const
- HWLOC_DECLSPEC hwloc_obj_type_t hwloc_obj_type_of_string (const char *string) __hwloc_attribute_pure
- HWLOC_DECLSPEC int hwloc_obj_type_snprintf (char *__hwloc_restrict string, size_t size, hwloc_obj_t obj, int verbose)
- HWLOC_DECLSPEC int hwloc_obj_attr_snprintf (char *__hwloc_restrict string, size_t size, hwloc_obj_t obj, const char *__hwloc_restrict separator, int verbose)
- HWLOC_DECLSPEC int hwloc_obj_snprintf (char *__hwloc_restrict string, size_t size, hwloc_topology_t topology, hwloc_obj_t obj, const char *__hwloc_restrict indexprefix, int verbose)
- HWLOC_DECLSPEC int hwloc_obj_cpuset_snprintf (char *__hwloc_restrict str, size_t size, size_t nobj, const hwloc_obj_t *__hwloc_restrict objs)
- static __hwloc_inline const char * hwloc_obj_get_info_by_name (hwloc_obj_t obj, const char *name) __hwloc_attribute_pure
- HWLOC_DECLSPEC void hwloc_obj_add_info (hwloc_obj_t obj, const char *name, const char *value)

15.11.1 Detailed Description

15.11.2 Function Documentation

15.11.2.1 HWLOC_DECLSPEC 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.

15.11.2.2 HWLOC_DECLSPEC int hwloc_obj_attr_snprintf (char *__hwloc_restrict *string*, size_t *size*, hwloc_obj_t *obj*, const char *__hwloc_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).

15.11.2.3 HWLOC_DECLSPEC int hwloc_obj_cpuset_snprintf (char *__hwloc_restrict *str*, size_t *size*, size_t *nobj*, const hwloc_obj_t *__hwloc_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).

15.11.2.4 static __hwloc_inline const char * hwloc_obj_get_info_by_name (hwloc_obj_t *obj*, const char * *name*) [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.

15.11.2.5 HWLOC_DECLSPEC int hwloc_obj_snprintf (char *__hwloc_restrict *string*, size_t *size*, hwloc_topology_t *topology*, hwloc_obj_t *obj*, const char *__hwloc_restrict *indexprefix*, int *verbose*)

Stringify a given topology object into a human-readable form.

Note

This function is deprecated in favor of hwloc_obj_type_snprintf() and hwloc_obj_attr_snprintf() since it is not very flexible and only prints physical/OS indexes.

Fill string `string` up to `size` characters with the description of topology object `obj` in topology `topology`.

If `verbose` is set, a longer description is used. Otherwise a short description is used.

`indexprefix` is used to prefix the `os_index` attribute number of the object in the description. If `NULL`, the `#` character is used.

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).

15.11.2.6 HWLOC_DECLSPEC hwloc_obj_type_t hwloc_obj_type_of_string (const char * *string*)

Return an object type from the string.

Returns

-1 if unrecognized.

15.11.2.7 HWLOC_DECLSPEC int hwloc_obj_type_snprintf (char *__hwloc_restrict *string*, size_t *size*, hwloc_obj_t *obj*, int *verbose*)

Stringify the type of a given topology object into a human-readable form.

It differs from hwloc_obj_type_string() because it prints type attributes such as cache depth.

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).

15.11.2.8 HWLOC_DECLSPEC const char* hwloc_obj_type_string (hwloc_obj_type_t *type*) const

Return a stringified topology object type.

15.12 CPU binding

Enumerations

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

Functions

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

15.12.1 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. Some operating systems also only support that kind of binding.

Note

Some operating systems do not provide all hwloc-supported mechanisms to bind processes, threads, etc. and the corresponding binding functions may fail. -1 is returned and 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).

The most portable version that should be preferred over the others, whenever possible, is

```
hwloc_set_cpubind(topology, set, 0),
```

as it just binds the current program, assuming it is single-threaded, or

```
hwloc_set_cpubind(topology, set, HWLOC_CPUBIND_THREAD),
```

which binds the current thread of the current program (which may be multithreaded).

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 can be a very convenient tool to check how binding actually happened.

15.12.2 Enumeration Type Documentation

15.12.2.1 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.

15.12.3 Function Documentation

15.12.3.1 HWLOC_DECLSPEC 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.

15.12.3.2 HWLOC_DECLSPEC 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.

15.12.3.3 HWLOC_DECLSPEC 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.
HWLOC_CPUBIND_THREAD can not be used in `flags`.
As a special case on Linux, if a tid (thread ID) is supplied instead of a pid (process ID), the binding for that specific thread is returned.

15.12.3.4 HWLOC_DECLSPEC 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.
HWLOC_CPUBIND_THREAD can not be used in `flags`.
As a special case on Linux, if a tid (thread ID) is supplied instead of a pid (process ID), the binding for that specific thread is returned.

15.12.3.5 HWLOC_DECLSPEC 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`.

15.12.3.6 HWLOC_DECLSPEC 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

15.12.3.7 HWLOC_DECLSPEC 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. HWLOC_CPUBIND_THREAD can not be used in flags.

15.12.3.8 HWLOC_DECLSPEC 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.

15.13 Memory binding

Enumerations

- enum hwloc_membind_policy_t {
  HWLOC_MEMBIND_DEFAULT, HWLOC_MEMBIND_FIRSTTOUCH, HWLOC_MEMBIND_BIND, HWLOC_MEMBIND_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_MEMBIND_MIGRATE,
  HWLOC_MEMBIND_NOCPUBIND }

Functions

- HWLOC_DECLSPEC int hwloc_set_membind_nodeset (hwloc_topology_t topology, hwloc_const_nodeset_t nodeset, hwloc_membind_policy_t policy, int flags)
- HWLOC_DECLSPEC int hwloc_set_membind (hwloc_topology_t topology, hwloc_const_cpuset_t cpuset, hwloc_membind_policy_t policy, int flags)
- HWLOC_DECLSPEC int hwloc_get_membind_nodeset (hwloc_topology_t topology, hwloc_nodeset_t nodeset, hwloc_membind_policy_t *policy, int flags)
- HWLOC_DECLSPEC int hwloc_get_membind (hwloc_topology_t topology, hwloc_cpuset_t cpuset, hwloc_membind_policy_t *policy, int flags)
- HWLOC_DECLSPEC 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)
- HWLOC_DECLSPEC int hwloc_set_proc_membind (hwloc_topology_t topology, hwloc_pid_t pid, hwloc_const_cpuset_t cpuset, hwloc_membind_policy_t policy, int flags)
- HWLOC_DECLSPEC 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)
- HWLOC_DECLSPEC int hwloc_get_proc_membind (hwloc_topology_t topology, hwloc_pid_t pid, hwloc_cpuset_t cpuset, hwloc_membind_policy_t *policy, int flags)
- HWLOC_DECLSPEC 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)
- HWLOC_DECLSPEC int hwloc_set_area_membind (hwloc_topology_t topology, const void *addr, size_t len, hwloc_const_cpuset_t cpuset, hwloc_membind_policy_t policy, int flags)
- HWLOC_DECLSPEC 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)
- HWLOC_DECLSPEC int hwloc_get_area_membind (hwloc_topology_t topology, const void *addr, size_t len, hwloc_cpuset_t cpuset, hwloc_membind_policy_t *policy, int flags)
- HWLOC_DECLSPEC void * hwloc_alloc (hwloc_topology_t topology, size_t len)
- HWLOC_DECLSPEC void * hwloc_alloc_membind_nodeset (hwloc_topology_t topology, size_t len, hwloc_const_nodeset_t nodeset, hwloc_membind_policy_t policy, int flags) __hwloc_attribute_malloc
- HWLOC_DECLSPEC void * hwloc_alloc_membind (hwloc_topology_t topology, size_t len, hwloc_const_cpuset_t cpuset, hwloc_membind_policy_t policy, int flags) __hwloc_attribute_malloc
- HWLOC_DECLSPEC int hwloc_free (hwloc_topology_t topology, void *addr, size_t len)

15.13.1 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.

Note

Not all operating systems support all three ways Using a binding flag or policy that is not supported by the underlying OS will cause hwloc's binding functions to fail and return -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 cpuset can not be enforced (e.g., some systems only allow binding memory to a single NUMA node).

The most portable form that should be preferred over the others whenever possible is as follows:

```
hwloc_alloc_membind_policy(topology, size, set,
                           HWLOC_MEMBIND_DEFAULT, 0);
```

This will allocate 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. Note that since HWLOC_MEMBIND_STRICT was not specified, failures to bind will not be reported – generally, only memory allocation failures will be reported (e.g., even a plain malloc() would have failed with ENOMEM).

Each hwloc memory binding function is available in two forms: one that takes a CPU set argument and another that takes a NUMA memory node set argument (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). The names of the latter form end with _nodeset. It is also possible to convert between CPU set and node set using hwloc_cpuset_to_nodeset() or hwloc_cpuset_from_nodeset().

Note

On some operating systems, memory binding affects the CPU binding; see HWLOC_MEMBIND_NOCPUBIND

15.13.2 Enumeration Type Documentation

15.13.2.1 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.

Note

Not all systems support all kinds of binding. 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.

15.13.2.2 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).

Note

Not all systems support all kinds of binding. 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.

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 cpuset 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 hwloc_get_membind∗() functions when multiple threads or parts of a memory area have differing memory binding policies.

15.13.3 Function Documentation

15.13.3.1 HWLOC_DECLSPEC 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().

15.13.3.2 HWLOC_DECLSPEC void* hwloc_alloc_membind (hwloc_topology_t *topology*, size_t *len*, hwloc_const_cpuset_t *cpuset*, hwloc_membind_policy_t *policy*, int *flags*)

Allocate some memory on memory nodes near the given physical cpuset `cpuset`.

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

Note

The allocated memory should be freed with hwloc_free().

15.13.3.3 HWLOC_DECLSPEC 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 the given physical nodeset `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

Note

The allocated memory should be freed with hwloc_free().

15.13.3.4 HWLOC_DECLSPEC 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().

15.13.3.5 HWLOC_DECLSPEC int hwloc_get_area_membind (hwloc_topology_t *topology*, const void * *addr*, size_t *len*, hwloc_cpuset_t *cpuset*, 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: `cpuset` 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 policy is returned in `policy`. `cpuset` is set to the union of CPUs near the NUMA node(s) in the nodeset.

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

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

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

15.13.3.7 HWLOC_DECLSPEC int hwloc_get_membind (hwloc_topology_t *topology,* hwloc_cpuset_t *cpuset,* hwloc_membind_policy_t * *policy,* int *flags*)

Query the default memory binding policy and physical locality of the current process or thread (the locality is returned in `cpuset` as CPUs near the locality's actual NUMA node(s)).

This function has two output parameters: `cpuset` 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 policy is returned in `policy`. `cpuset` is set to the union of CPUs near the NUMA node(s) in the nodeset.

Otherwise, if HWLOC_MEMBIND_PROCESS is specified (and HWLOC_MEMBIND_STRICT is *not* specified), the default nodeset from each thread is logically OR'ed together. `cpuset` is set to the union of CPUs near the NUMA node(s) in the resulting 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. The policy is returned in `policy`; `cpuset` is set to the union of CPUs near the NUMA node(s) in the `nodeset`.

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

15.13.3.8 HWLOC_DECLSPEC 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.

15.13.3.9 HWLOC_DECLSPEC int hwloc_get_proc_membind (hwloc_topology_t *topology,* hwloc_pid_t *pid,* hwloc_cpuset_t *cpuset,* hwloc_membind_policy_t * *policy,* int *flags*)

Query the default memory binding policy and physical locality of the specified process (the locality is returned in `cpuset` as CPUs near the locality's actual NUMA node(s)).

This function has two output parameters: `cpuset` 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 policy is returned in `policy`. `cpuset` is set to the union of CPUs near the NUMA node(s) in the nodeset.

Otherwise, the default nodeset from each thread is logically OR'ed together. `cpuset` is set to the union of CPUs near the NUMA node(s) in the resulting 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.

15.13.3.10 HWLOC_DECLSPEC 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.

15.13.3.11 HWLOC_DECLSPEC int hwloc_set_area_membind (hwloc_topology_t *topology*, const void * *addr*, size_t *len*, hwloc_const_cpuset_t *cpuset*, hwloc_membind_policy_t *policy*, int *flags*)

Bind the already-allocated memory identified by (addr, len) to the NUMA node(s) near physical 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

15.13.3.12 HWLOC_DECLSPEC 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) in physical 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

15.13.3.13 HWLOC_DECLSPEC int hwloc_set_membind (hwloc_topology_t *topology,* hwloc_const_cpuset_t *cpuset,* 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) near the specified physical `cpuset`.

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

15.13.3.14 HWLOC_DECLSPEC 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 physical `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

15.13.3.15 HWLOC_DECLSPEC int hwloc_set_proc_membind (hwloc_topology_t *topology,* hwloc_pid_t *pid,* hwloc_const_cpuset_t *cpuset,* hwloc_membind_policy_t *policy,* int *flags*)

Set the default memory binding policy of the specified process to prefer the NUMA node(s) near the specified physical `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.

15.13.3.16 HWLOC_DECLSPEC 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 physical `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.

15.14 Building Custom Topologies

Functions

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

15.14.1 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.

15.14.2 Function Documentation

15.14.2.1 HWLOC_DECLSPEC 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().

15.14.2.2 HWLOC_DECLSPEC 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().

15.15 Object Type Helpers

Functions

- static __hwloc_inline int hwloc_get_type_or_below_depth (hwloc_topology_t topology, hwloc_obj_type_t type) __hwloc_attribute_pure
- static __hwloc_inline int hwloc_get_type_or_above_depth (hwloc_topology_t topology, hwloc_obj_type_t type) __hwloc_attribute_pure

15.15.1 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 socket has fewer caches than its peers.

15.15.2 Function Documentation

15.15.2.1 static __hwloc_inline int hwloc_get_type_or_above_depth (hwloc_topology_t *topology,* hwloc_obj_type_t *type*) [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.

15.15.2.2 static __hwloc_inline int hwloc_get_type_or_below_depth (hwloc_topology_t *topology,* hwloc_obj_type_t *type*) [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.

15.16 Basic Traversal Helpers

Functions

- static __hwloc_inline hwloc_obj_t hwloc_get_root_obj (hwloc_topology_t topology) __hwloc_attribute_pure
- static __hwloc_inline hwloc_obj_t hwloc_get_ancestor_obj_by_depth (hwloc_topology_t topology __hwloc_attribute_unused, unsigned depth, hwloc_obj_t obj) __hwloc_attribute_pure
- static __hwloc_inline hwloc_obj_t hwloc_get_ancestor_obj_by_type (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_obj_type_t type, hwloc_obj_t obj) __hwloc_attribute_pure
- static __hwloc_inline hwloc_obj_t hwloc_get_next_obj_by_depth (hwloc_topology_t topology, unsigned depth, hwloc_obj_t prev)
- static __hwloc_inline hwloc_obj_t hwloc_get_next_obj_by_type (hwloc_topology_t topology, hwloc_obj_type_t type, hwloc_obj_t prev)
- static __hwloc_inline hwloc_obj_t hwloc_get_pu_obj_by_os_index (hwloc_topology_t topology, unsigned os_index) __hwloc_attribute_pure
- static __hwloc_inline hwloc_obj_t hwloc_get_next_child (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_obj_t parent, hwloc_obj_t prev)
- static __hwloc_inline hwloc_obj_t hwloc_get_common_ancestor_obj (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_obj_t obj1, hwloc_obj_t obj2) __hwloc_attribute_pure
- static __hwloc_inline int hwloc_obj_is_in_subtree (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_obj_t obj, hwloc_obj_t subtree_root) __hwloc_attribute_pure

15.16.1 Detailed Description

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

15.16.2 Function Documentation

15.16.2.1 static __hwloc_inline hwloc_obj_t hwloc_get_ancestor_obj_by_depth (hwloc_topology_t topology __hwloc_attribute_unused, unsigned *depth*, hwloc_obj_t *obj*) [static]

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

15.16.2.2 static __hwloc_inline hwloc_obj_t hwloc_get_ancestor_obj_by_type (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_obj_type_t *type*, hwloc_obj_t *obj*) [static]

Returns the ancestor object of `obj` with type `type`.

15.16.2.3 static __hwloc_inline hwloc_obj_t hwloc_get_common_ancestor_obj (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_obj_t *obj1*, hwloc_obj_t *obj2*) [static]

Returns the common parent object to objects lvl1 and lvl2.

15.16.2.4 static __hwloc_inline hwloc_obj_t hwloc_get_next_child (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_obj_t *parent*, hwloc_obj_t *prev*) [static]

Return the next child.

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

15.16.2.5 static __hwloc_inline hwloc_obj_t hwloc_get_next_obj_by_depth (hwloc_topology_t *topology,* unsigned *depth,* hwloc_obj_t *prev*) [static]

Returns the next object at depth `depth`.

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

15.16.2.6 static __hwloc_inline hwloc_obj_t hwloc_get_next_obj_by_type (hwloc_topology_t *topology,* hwloc_obj_type_t *type,* hwloc_obj_t *prev*) [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().

15.16.2.7 static __hwloc_inline hwloc_obj_t hwloc_get_pu_obj_by_os_index (hwloc_topology_t *topology,* unsigned *os_index*) [static]

Returns the object of type HWLOC_OBJ_PU with `os_index`.

Note

The `os_index` field of object should most of the times only be used for pretty-printing purpose. Type HWLOC_OBJ_PU is the only case where `os_index` could actually be useful, when manually binding to processors. However, using CPU sets to hide this complexity should often be preferred.

15.16.2.8 static __hwloc_inline hwloc_obj_t hwloc_get_root_obj (hwloc_topology_t *topology*) [static]

Returns the top-object of the topology-tree.

Its type is typically HWLOC_OBJ_MACHINE but it could be different for complex topologies. This function replaces the old deprecated hwloc_get_system_obj().

15.16.2.9 static __hwloc_inline int hwloc_obj_is_in_subtree (hwloc_topology_t topology __*hwloc_attribute_unused,* hwloc_obj_t *obj,* hwloc_obj_t *subtree_root*) [static]

Returns true if `obj` is inside the subtree beginning with `subtree_root`.

Note

This function assumes that both `obj` and `subtree_root` have a `cpuset`.

15.17 Finding Objects Inside a CPU set

Functions

- static __hwloc_inline hwloc_obj_t hwloc_get_first_largest_obj_inside_cpuset (hwloc_topology_t topology, hwloc_const_cpuset_t set)
- HWLOC_DECLSPEC int hwloc_get_largest_objs_inside_cpuset (hwloc_topology_t topology, hwloc_const_cpuset_t set, hwloc_obj_t *__hwloc_restrict objs, int max)
- static __hwloc_inline 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_inline 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_inline hwloc_obj_t hwloc_get_obj_inside_cpuset_by_depth (hwloc_topology_t topology, hwloc_const_cpuset_t set, unsigned depth, unsigned idx) __hwloc_attribute_pure
- static __hwloc_inline 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) __hwloc_attribute_pure
- static __hwloc_inline unsigned hwloc_get_nbobjs_inside_cpuset_by_depth (hwloc_topology_t topology, hwloc_const_cpuset_t set, unsigned depth) __hwloc_attribute_pure
- static __hwloc_inline int hwloc_get_nbobjs_inside_cpuset_by_type (hwloc_topology_t topology, hwloc_const_cpuset_t set, hwloc_obj_type_t type) __hwloc_attribute_pure
- static __hwloc_inline int hwloc_get_obj_index_inside_cpuset (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_const_cpuset_t set, hwloc_obj_t obj) __hwloc_attribute_pure

15.17.1 Detailed Description

15.17.2 Function Documentation

15.17.2.1 static __hwloc_inline hwloc_obj_t hwloc_get_first_largest_obj_inside_cpuset (hwloc_topology_t *topology,* hwloc_const_cpuset_t *set*) [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.

15.17.2.2 HWLOC_DECLSPEC int hwloc_get_largest_objs_inside_cpuset (hwloc_topology_t *topology,* hwloc_const_cpuset_t *set,* hwloc_obj_t *__hwloc_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.

15.17.2.3 static __hwloc_inline unsigned hwloc_get_nbobjs_inside_cpuset_by_depth (hwloc_topology_t *topology,* hwloc_const_cpuset_t *set,* unsigned *depth*) [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.

15.17.2.4 static __hwloc_inline int hwloc_get_nbobjs_inside_cpuset_by_type (hwloc_topology_t *topology,* hwloc_const_cpuset_t *set,* hwloc_obj_type_t *type*) [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.

15.17.2.5 static __hwloc_inline 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]

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.

15.17.2.6 static __hwloc_inline 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]

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.

15.17.2.7 static __hwloc_inline int hwloc_get_obj_index_inside_cpuset (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_const_cpuset_t *set*, hwloc_obj_t *obj*) [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`.

15.17.2.8 static __hwloc_inline hwloc_obj_t hwloc_get_obj_inside_cpuset_by_depth (hwloc_topology_t *topology*, hwloc_const_cpuset_t *set*, unsigned *depth*, unsigned *idx*) [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.

15.17.2.9 static __hwloc_inline 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]

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.

15.18 Finding a single Object covering at least CPU set

Functions

- static __hwloc_inline hwloc_obj_t hwloc_get_child_covering_cpuset (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_const_cpuset_t set, hwloc_obj_t parent) __hwloc_attribute_pure
- static __hwloc_inline hwloc_obj_t hwloc_get_obj_covering_cpuset (hwloc_topology_t topology, hwloc_const_cpuset_t set) __hwloc_attribute_pure

15.18.1 Detailed Description

15.18.2 Function Documentation

15.18.2.1 static __hwloc_inline **hwloc_obj_t** hwloc_get_child_covering_cpuset (**hwloc_topology_t** topology __*hwloc_attribute_unused,* **hwloc_const_cpuset_t** *set,* **hwloc_obj_t** *parent*) [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.

15.18.2.2 static __hwloc_inline **hwloc_obj_t** hwloc_get_obj_covering_cpuset (**hwloc_topology_t** *topology,* **hwloc_const_cpuset_t** *set*) [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.

15.19 Finding a set of similar Objects covering at least a CPU set

Functions

- static __hwloc_inline 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_inline 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)

15.19.1 Detailed Description

15.19.2 Function Documentation

15.19.2.1 static __hwloc_inline 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]

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.

15.19.2.2 static __hwloc_inline 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*) [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.

15.20 Cache-specific Finding Helpers

Functions

- static __hwloc_inline hwloc_obj_t hwloc_get_cache_covering_cpuset (hwloc_topology_t topology, hwloc_const_cpuset_t set) __hwloc_attribute_pure
- static __hwloc_inline hwloc_obj_t hwloc_get_shared_cache_covering_obj (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_obj_t obj) __hwloc_attribute_pure

15.20.1 Detailed Description

15.20.2 Function Documentation

15.20.2.1 static __hwloc_inline hwloc_obj_t hwloc_get_cache_covering_cpuset (hwloc_topology_t *topology,* hwloc_const_cpuset_t *set*) [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.

15.20.2.2 static __hwloc_inline hwloc_obj_t hwloc_get_shared_cache_covering_obj (hwloc_topology_t topology __*hwloc_attribute_unused,* hwloc_obj_t *obj*) [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.

15.21 Advanced Traversal Helpers

Functions

- HWLOC_DECLSPEC unsigned hwloc_get_closest_objs (hwloc_topology_t topology, hwloc_obj_t src, hwloc_obj_t *__hwloc_restrict objs, unsigned max)
- static __hwloc_inline 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) __hwloc_attribute_pure
- static __hwloc_inline hwloc_obj_t hwloc_get_obj_below_array_by_type (hwloc_topology_t topology, int nr, hwloc_obj_type_t *typev, unsigned *idxv) __hwloc_attribute_pure

15.21.1 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 socket has fewer caches than its peers.

15.21.2 Function Documentation

15.21.2.1 HWLOC_DECLSPEC unsigned hwloc_get_closest_objs (hwloc_topology_t *topology,* hwloc_obj_t *src,* hwloc_obj_t *__hwloc_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.

15.21.2.2 static __hwloc_inline hwloc_obj_t hwloc_get_obj_below_array_by_type (hwloc_topology_t *topology,* int *nr,* hwloc_obj_type_t * *typev,* unsigned * *idxv*) [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, SOCKET and CORE, and idxv contains 0, 1 and 2, return the third core object below the second socket below the first NUMA node.

Note

This function requires all these objects and the root object to have a CPU set.

15.21.2.3 static __hwloc_inline 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]

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 SOCKET, idx1 is 2, type2 is CORE and idx2 is 3, return the fourth core object below the third socket.

Note

This function requires these objects to have a CPU set.

15.22 Binding Helpers

Functions

- static __hwloc_inline void hwloc_distributev (hwloc_topology_t topology, hwloc_obj_t *root, unsigned n_roots, hwloc_cpuset_t *cpuset, unsigned n, unsigned until)
- static __hwloc_inline void hwloc_distribute (hwloc_topology_t topology, hwloc_obj_t root, hwloc_cpuset_t *cpuset, unsigned n, unsigned until)
- static __hwloc_inline 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 __hwloc_inline void * hwloc_alloc_membind_policy (hwloc_topology_t topology, size_t len, hwloc_const_cpuset_t cpuset, hwloc_membind_policy_t policy, int flags)

15.22.1 Detailed Description

15.22.2 Function Documentation

15.22.2.1 static __hwloc_inline void* hwloc_alloc_membind_policy (hwloc_topology_t *topology,* size_t *len,* hwloc_const_cpuset_t *cpuset,* hwloc_membind_policy_t *policy,* int *flags*) [static]

Allocate some memory on the memory nodes near given cpuset `cpuset`.

This is similar to hwloc_alloc_membind_policy_nodeset, but for a given cpuset.

15.22.2.2 static __hwloc_inline 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]

Allocate some memory on the given nodeset `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.

15.22.2.3 static __hwloc_inline void hwloc_distribute (hwloc_topology_t *topology,* hwloc_obj_t *root,* hwloc_cpuset_t * *cpuset,* unsigned *n,* unsigned *until*) [static]

15.22.2.4 static __hwloc_inline void hwloc_distributev (hwloc_topology_t *topology,* hwloc_obj_t * *roots,* unsigned *n_roots,* hwloc_cpuset_t * *cpuset,* unsigned *n,* unsigned *until*) [static]

Distribute `n` items over the topology under `root`.

Distribute `n` items over the topology under `roots`.

Array `cpuset` will be filled with `n` cpusets recursively distributed linearly over the topology under `root`, down to depth `until` (which can be INT_MAX to distribute down to the finest level).

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.

Note

This function requires the `root` object to have a CPU set.

This is the same as hwloc_distribute, but takes an array of roots instead of just one root.

Note

This function requires the `roots` objects to have a CPU set.

15.23 Cpuset Helpers

Functions

- static __hwloc_inline hwloc_const_cpuset_t hwloc_topology_get_complete_cpuset (hwloc_topology_t topology) __hwloc_attribute_pure
- static __hwloc_inline hwloc_const_cpuset_t hwloc_topology_get_topology_cpuset (hwloc_topology_t topology) __hwloc_attribute_pure
- static __hwloc_inline hwloc_const_cpuset_t hwloc_topology_get_online_cpuset (hwloc_topology_t topology) __hwloc_attribute_pure
- static __hwloc_inline hwloc_const_cpuset_t hwloc_topology_get_allowed_cpuset (hwloc_topology_t topology) __hwloc_attribute_pure

15.23.1 Detailed Description

15.23.2 Function Documentation

15.23.2.1 static __hwloc_inline hwloc_const_cpuset_t hwloc_topology_get_allowed_cpuset (hwloc_topology_t *topology*) [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_cpuset_dup must be used to obtain a local copy.

15.23.2.2 static __hwloc_inline hwloc_const_cpuset_t hwloc_topology_get_complete_cpuset (hwloc_topology_t *topology*) [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_cpuset_dup must be used to obtain a local copy.

15.23.2.3 static __hwloc_inline hwloc_const_cpuset_t hwloc_topology_get_online_cpuset (hwloc_topology_t *topology*) [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_cpuset_dup must be used to obtain a local copy.

15.23.2.4 static __hwloc_inline hwloc_const_cpuset_t hwloc_topology_get_topology_cpuset (hwloc_topology_t *topology*) [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_cpuset_dup must be used to obtain a local copy.

15.24 Nodeset Helpers

Functions

- static __hwloc_inline hwloc_const_nodeset_t hwloc_topology_get_complete_nodeset (hwloc_topology_t topology) __hwloc_attribute_pure
- static __hwloc_inline hwloc_const_nodeset_t hwloc_topology_get_topology_nodeset (hwloc_topology_t topology) __hwloc_attribute_pure
- static __hwloc_inline hwloc_const_nodeset_t hwloc_topology_get_allowed_nodeset (hwloc_topology_t topology) __hwloc_attribute_pure

15.24.1 Detailed Description

15.24.2 Function Documentation

15.24.2.1 static __hwloc_inline hwloc_const_nodeset_t hwloc_topology_get_allowed_nodeset (hwloc_topology_t *topology*) [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_nodeset_dup must be used to obtain a local copy.

15.24.2.2 static __hwloc_inline hwloc_const_nodeset_t hwloc_topology_get_complete_nodeset (hwloc_topology_t *topology*) [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_nodeset_dup must be used to obtain a local copy.

15.24.2.3 static __hwloc_inline hwloc_const_nodeset_t hwloc_topology_get_topology_nodeset (hwloc_topology_t *topology*) [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_nodeset_dup must be used to obtain a local copy.

15.25 Conversion between cpuset and nodeset

Functions

- static __hwloc_inline void hwloc_cpuset_to_nodeset (hwloc_topology_t topology, hwloc_const_cpuset_t cpuset, hwloc_nodeset_t nodeset)
- static __hwloc_inline void hwloc_cpuset_to_nodeset_strict (struct hwloc_topology *topology, hwloc_const_cpuset_t cpuset, hwloc_nodeset_t nodeset)
- static __hwloc_inline void hwloc_cpuset_from_nodeset (hwloc_topology_t topology, hwloc_cpuset_t cpuset, hwloc_const_nodeset_t nodeset)
- static __hwloc_inline void hwloc_cpuset_from_nodeset_strict (struct hwloc_topology *topology, hwloc_cpuset_t cpuset, hwloc_const_nodeset_t nodeset)

15.25.1 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-NUMA machines, the strict conversion routines may be used instead.

15.25.2 Function Documentation

15.25.2.1 static __hwloc_inline void hwloc_cpuset_from_nodeset (hwloc_topology_t *topology,* hwloc_cpuset_t *cpuset,* hwloc_const_nodeset_t *nodeset*) [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.

15.25.2.2 static __hwloc_inline void hwloc_cpuset_from_nodeset_strict (struct hwloc_topology * *topology,* hwloc_cpuset_t *cpuset,* hwloc_const_nodeset_t *nodeset*) [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.

15.25.2.3 static __hwloc_inline void hwloc_cpuset_to_nodeset (hwloc_topology_t *topology,* hwloc_const_cpuset_t *cpuset,* hwloc_nodeset_t *nodeset*) [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.

15.25.2.4 static __hwloc_inline void hwloc_cpuset_to_nodeset_strict (struct hwloc_topology * *topology,* hwloc_const_cpuset_t *cpuset,* hwloc_nodeset_t *nodeset*) [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.

15.26 Distances

Functions

- static __hwloc_inline struct hwloc_distances_s * hwloc_get_whole_distance_matrix_by_depth (hwloc_topology_t topology, unsigned depth)
- static __hwloc_inline struct hwloc_distances_s * hwloc_get_whole_distance_matrix_by_type (hwloc_topology_t topology, hwloc_obj_type_t type)
- static __hwloc_inline 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 __hwloc_inline int hwloc_get_latency (hwloc_topology_t topology, hwloc_obj_t obj1, hwloc_obj_t obj2, float *latency, float *reverse_latency)

15.26.1 Detailed Description

15.26.2 Function Documentation

15.26.2.1 static __hwloc_inline 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], [read]

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.

15.26.2.2 static __hwloc_inline int hwloc_get_latency (hwloc_topology_t *topology,* hwloc_obj_t *obj1,* hwloc_obj_t *obj2,* float * *latency,* float * *reverse_latency*) [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.

15.26.2.3 static __hwloc_inline struct hwloc_distances_s* hwloc_get_whole_distance_matrix_by_depth (hwloc_topology_t *topology,* unsigned *depth*) [static], [read]

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.

15.26.2.4 static __hwloc_inline struct **hwloc_distances_s**∗ hwloc_get_whole_distance_matrix_by_type (**hwloc_topology_t** *topology,* **hwloc_obj_type_t** *type*) [static],[read]

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.

15.27 Advanced I/O object traversal helpers

Functions

- static __hwloc_inline hwloc_obj_t hwloc_get_non_io_ancestor_obj (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_obj_t ioobj)
- static __hwloc_inline hwloc_obj_t hwloc_get_next_pcidev (hwloc_topology_t topology, hwloc_obj_t prev)
- static __hwloc_inline hwloc_obj_t hwloc_get_pcidev_by_busid (hwloc_topology_t topology, unsigned domain, unsigned bus, unsigned dev, unsigned func)
- static __hwloc_inline hwloc_obj_t hwloc_get_pcidev_by_busidstring (hwloc_topology_t topology, const char *busid)
- static __hwloc_inline hwloc_obj_t hwloc_get_next_osdev (hwloc_topology_t topology, hwloc_obj_t prev)
- static __hwloc_inline hwloc_obj_t hwloc_get_next_bridge (hwloc_topology_t topology, hwloc_obj_t prev)
- static __hwloc_inline int hwloc_bridge_covers_pcibus (hwloc_obj_t bridge, unsigned domain, unsigned bus)
- static __hwloc_inline hwloc_obj_t hwloc_get_hostbridge_by_pcibus (hwloc_topology_t topology, unsigned domain, unsigned bus)

15.27.1 Detailed Description

15.27.2 Function Documentation

15.27.2.1 static __hwloc_inline int hwloc_bridge_covers_pcibus (hwloc_obj_t *bridge*, unsigned *domain*, unsigned *bus*) [static]

15.27.2.2 static __hwloc_inline hwloc_obj_t hwloc_get_hostbridge_by_pcibus (hwloc_topology_t *topology*, unsigned *domain*, unsigned *bus*) [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.

15.27.2.3 static __hwloc_inline hwloc_obj_t hwloc_get_next_bridge (hwloc_topology_t *topology*, hwloc_obj_t *prev*) [static]

Get the next bridge in the system.

Returns

the first bridge if `prev` is `NULL`.

15.27.2.4 static __hwloc_inline hwloc_obj_t hwloc_get_next_osdev (hwloc_topology_t *topology*, hwloc_obj_t *prev*) [static]

Get the next OS device in the system.

Returns

the first OS device if `prev` is `NULL`.

15.27.2.5 static __hwloc_inline hwloc_obj_t hwloc_get_next_pcidev (hwloc_topology_t *topology,* hwloc_obj_t *prev*) [static]

Get the next PCI device in the system.

Returns

the first PCI device if `prev` is `NULL`.

15.27.2.6 static __hwloc_inline hwloc_obj_t hwloc_get_non_io_ancestor_obj (hwloc_topology_t topology __*hwloc_attribute_unused,* hwloc_obj_t *ioobj*) [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`.

15.27.2.7 static __hwloc_inline hwloc_obj_t hwloc_get_pcidev_by_busid (hwloc_topology_t *topology,* unsigned *domain,* unsigned *bus,* unsigned *dev,* unsigned *func*) [static]

Find the PCI device object matching the PCI bus id given domain, bus device and function PCI bus id.

15.27.2.8 static __hwloc_inline hwloc_obj_t hwloc_get_pcidev_by_busidstring (hwloc_topology_t *topology,* const char * *busid*) [static]

Find the PCI device object matching the PCI bus id given as a string xxxx:yy:zz.t or yy:zz.t.

15.28 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 struct hwloc_bitmap_s * hwloc_const_bitmap_t

Functions

- HWLOC_DECLSPEC hwloc_bitmap_t hwloc_bitmap_alloc (void) __hwloc_attribute_malloc
- HWLOC_DECLSPEC hwloc_bitmap_t hwloc_bitmap_alloc_full (void) __hwloc_attribute_malloc
- HWLOC_DECLSPEC void hwloc_bitmap_free (hwloc_bitmap_t bitmap)
- HWLOC_DECLSPEC hwloc_bitmap_t hwloc_bitmap_dup (hwloc_const_bitmap_t bitmap) __hwloc_attribute_malloc
- HWLOC_DECLSPEC void hwloc_bitmap_copy (hwloc_bitmap_t dst, hwloc_const_bitmap_t src)
- HWLOC_DECLSPEC int hwloc_bitmap_snprintf (char *__hwloc_restrict buf, size_t buflen, hwloc_const_bitmap_t bitmap)
- HWLOC_DECLSPEC int hwloc_bitmap_asprintf (char **strp, hwloc_const_bitmap_t bitmap)
- HWLOC_DECLSPEC int hwloc_bitmap_sscanf (hwloc_bitmap_t bitmap, const char *__hwloc_restrict string)
- HWLOC_DECLSPEC int hwloc_bitmap_list_snprintf (char *__hwloc_restrict buf, size_t buflen, hwloc_const_bitmap_t bitmap)
- HWLOC_DECLSPEC int hwloc_bitmap_list_asprintf (char **strp, hwloc_const_bitmap_t bitmap)
- HWLOC_DECLSPEC int hwloc_bitmap_list_sscanf (hwloc_bitmap_t bitmap, const char *__hwloc_restrict string)
- HWLOC_DECLSPEC int hwloc_bitmap_taskset_snprintf (char *__hwloc_restrict buf, size_t buflen, hwloc_const_bitmap_t bitmap)
- HWLOC_DECLSPEC int hwloc_bitmap_taskset_asprintf (char **strp, hwloc_const_bitmap_t bitmap)
- HWLOC_DECLSPEC int hwloc_bitmap_taskset_sscanf (hwloc_bitmap_t bitmap, const char *__hwloc_restrict string)
- HWLOC_DECLSPEC void hwloc_bitmap_zero (hwloc_bitmap_t bitmap)
- HWLOC_DECLSPEC void hwloc_bitmap_fill (hwloc_bitmap_t bitmap)
- HWLOC_DECLSPEC void hwloc_bitmap_only (hwloc_bitmap_t bitmap, unsigned id)
- HWLOC_DECLSPEC void hwloc_bitmap_allbut (hwloc_bitmap_t bitmap, unsigned id)
- HWLOC_DECLSPEC void hwloc_bitmap_from_ulong (hwloc_bitmap_t bitmap, unsigned long mask)
- HWLOC_DECLSPEC void hwloc_bitmap_from_ith_ulong (hwloc_bitmap_t bitmap, unsigned i, unsigned long mask)
- HWLOC_DECLSPEC void hwloc_bitmap_set (hwloc_bitmap_t bitmap, unsigned id)
- HWLOC_DECLSPEC void hwloc_bitmap_set_range (hwloc_bitmap_t bitmap, unsigned begin, int end)
- HWLOC_DECLSPEC void hwloc_bitmap_set_ith_ulong (hwloc_bitmap_t bitmap, unsigned i, unsigned long mask)
- HWLOC_DECLSPEC void hwloc_bitmap_clr (hwloc_bitmap_t bitmap, unsigned id)
- HWLOC_DECLSPEC void hwloc_bitmap_clr_range (hwloc_bitmap_t bitmap, unsigned begin, int end)
- HWLOC_DECLSPEC void hwloc_bitmap_singlify (hwloc_bitmap_t bitmap)
- HWLOC_DECLSPEC unsigned long hwloc_bitmap_to_ulong (hwloc_const_bitmap_t bitmap) __hwloc_attribute_pure

- HWLOC_DECLSPEC unsigned long hwloc_bitmap_to_ith_ulong (hwloc_const_bitmap_t bitmap, unsigned i) __hwloc_attribute_pure
- HWLOC_DECLSPEC int hwloc_bitmap_isset (hwloc_const_bitmap_t bitmap, unsigned id) __hwloc_attribute_pure
- HWLOC_DECLSPEC int hwloc_bitmap_iszero (hwloc_const_bitmap_t bitmap) __hwloc_attribute_pure
- HWLOC_DECLSPEC int hwloc_bitmap_isfull (hwloc_const_bitmap_t bitmap) __hwloc_attribute_pure
- HWLOC_DECLSPEC int hwloc_bitmap_first (hwloc_const_bitmap_t bitmap) __hwloc_attribute_pure
- HWLOC_DECLSPEC int hwloc_bitmap_next (hwloc_const_bitmap_t bitmap, int prev) __hwloc_attribute_pure
- HWLOC_DECLSPEC int hwloc_bitmap_last (hwloc_const_bitmap_t bitmap) __hwloc_attribute_pure
- HWLOC_DECLSPEC int hwloc_bitmap_weight (hwloc_const_bitmap_t bitmap) __hwloc_attribute_pure
- HWLOC_DECLSPEC void hwloc_bitmap_or (hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)
- HWLOC_DECLSPEC void hwloc_bitmap_and (hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)
- HWLOC_DECLSPEC void hwloc_bitmap_andnot (hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)
- HWLOC_DECLSPEC void hwloc_bitmap_xor (hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2)
- HWLOC_DECLSPEC void hwloc_bitmap_not (hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap)
- HWLOC_DECLSPEC int hwloc_bitmap_intersects (hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2) __hwloc_attribute_pure
- HWLOC_DECLSPEC int hwloc_bitmap_isincluded (hwloc_const_bitmap_t sub_bitmap, hwloc_const_bitmap_t super_bitmap) __hwloc_attribute_pure
- HWLOC_DECLSPEC int hwloc_bitmap_isequal (hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2) __hwloc_attribute_pure
- HWLOC_DECLSPEC int hwloc_bitmap_compare_first (hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2) __hwloc_attribute_pure
- HWLOC_DECLSPEC int hwloc_bitmap_compare (hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2) __hwloc_attribute_pure

15.28.1 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.

15.28.2 Macro Definition Documentation

15.28.2.1 #define hwloc_bitmap_foreach_begin(*id, bitmap*)

Loop macro iterating on bitmap `bitmap`.

`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 that 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.

15.28.2.2 #define hwloc_bitmap_foreach_end( )

Value:

```
} \
} while (0)
```

15.28.3 Typedef Documentation

15.28.3.1 typedef struct hwloc_bitmap_s* hwloc_bitmap_t

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

15.28.3.2 typedef struct hwloc_bitmap_s* hwloc_const_bitmap_t

a non-modifiable hwloc_bitmap_t

15.28.4 Function Documentation

15.28.4.1 HWLOC_DECLSPEC void hwloc_bitmap_allbut (hwloc_bitmap_t *bitmap,* unsigned *id*)

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

15.28.4.2 HWLOC_DECLSPEC 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().

15.28.4.3 HWLOC_DECLSPEC hwloc_bitmap_t hwloc_bitmap_alloc_full (void)

Allocate a new full bitmap.

15.28.4.4 HWLOC_DECLSPEC 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`.

15.28.4.5 HWLOC_DECLSPEC 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`.

15.28.4.6 HWLOC_DECLSPEC int hwloc_bitmap_asprintf (char ** *strp,* hwloc_const_bitmap_t *bitmap*)

Stringify a bitmap into a newly allocated string.

15.28.4.7 HWLOC_DECLSPEC void hwloc_bitmap_clr (hwloc_bitmap_t *bitmap,* unsigned *id*)

Remove index `id` from bitmap `bitmap`.

15.28.4.8 HWLOC_DECLSPEC 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.

15.28.4.9 HWLOC_DECLSPEC int hwloc_bitmap_compare (hwloc_const_bitmap_t *bitmap1,* hwloc_const_bitmap_t *bitmap2*)

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

Higher most significant bit is higher. The empty bitmap is considered lower than anything.

15.28.4.10 HWLOC_DECLSPEC 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.

15.28.4.11 HWLOC_DECLSPEC 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`.

15.28.4.12 HWLOC_DECLSPEC 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.

15.28.4.13 HWLOC_DECLSPEC 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)

15.28.4.14 HWLOC_DECLSPEC 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.

15.28.4.15 HWLOC_DECLSPEC void hwloc_bitmap_free (hwloc_bitmap_t *bitmap*)

Free bitmap `bitmap`.

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

15.28.4.16 HWLOC_DECLSPEC 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.

15.28.4.17 HWLOC_DECLSPEC void hwloc_bitmap_from_ulong (hwloc_bitmap_t *bitmap,* unsigned long *mask*)

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

15.28.4.18 HWLOC_DECLSPEC int hwloc_bitmap_intersects (hwloc_const_bitmap_t *bitmap1,* hwloc_const_bitmap_t *bitmap2*)

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

15.28.4.19 HWLOC_DECLSPEC int hwloc_bitmap_isequal (hwloc_const_bitmap_t *bitmap1,* hwloc_const_bitmap_t *bitmap2*)

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

15.28.4.20 HWLOC_DECLSPEC int hwloc_bitmap_isfull (hwloc_const_bitmap_t *bitmap*)

Test whether bitmap `bitmap` is completely full.

15.28.4.21 HWLOC_DECLSPEC 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`.

15.28.4.22 HWLOC_DECLSPEC int hwloc_bitmap_isset (hwloc_const_bitmap_t *bitmap,* unsigned *id*)

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

15.28.4.23 HWLOC_DECLSPEC int hwloc_bitmap_iszero (hwloc_const_bitmap_t *bitmap*)

Test whether bitmap `bitmap` is empty.

15.28.4.24 HWLOC_DECLSPEC 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.

15.28.4.25 HWLOC_DECLSPEC int hwloc_bitmap_list_asprintf (char ** *strp*, hwloc_const_bitmap_t *bitmap*)

Stringify a bitmap into a newly allocated list string.

15.28.4.26 HWLOC_DECLSPEC int hwloc_bitmap_list_snprintf (char *__hwloc_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).

15.28.4.27 HWLOC_DECLSPEC int hwloc_bitmap_list_sscanf (hwloc_bitmap_t *bitmap*, const char *__hwloc_restrict *string*)

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

15.28.4.28 HWLOC_DECLSPEC 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.

15.28.4.29 HWLOC_DECLSPEC void hwloc_bitmap_not (hwloc_bitmap_t *res*, hwloc_const_bitmap_t *bitmap*)

Negate bitmap `bitmap` and store the result in bitmap `res`.

15.28.4.30 HWLOC_DECLSPEC void hwloc_bitmap_only (hwloc_bitmap_t *bitmap*, unsigned *id*)

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

15.28.4.31 HWLOC_DECLSPEC 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`.

15.28.4.32 HWLOC_DECLSPEC void hwloc_bitmap_set (hwloc_bitmap_t *bitmap*, unsigned *id*)

Add index `id` in bitmap `bitmap`.

15.28.4.33 HWLOC_DECLSPEC 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`.

15.28.4.34 HWLOC_DECLSPEC 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.

15.28.4.35 HWLOC_DECLSPEC 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.

15.28.4.36 HWLOC_DECLSPEC int hwloc_bitmap_snprintf (char *__hwloc_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).

15.28.4.37 HWLOC_DECLSPEC int hwloc_bitmap_sscanf (hwloc_bitmap_t *bitmap,* const char *__hwloc_restrict *string*)

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

15.28.4.38 HWLOC_DECLSPEC int hwloc_bitmap_taskset_asprintf (char ** *strp,* hwloc_const_bitmap_t *bitmap*)

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

15.28.4.39 HWLOC_DECLSPEC int hwloc_bitmap_taskset_snprintf (char *__hwloc_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).

15.28.4.40 HWLOC_DECLSPEC int hwloc_bitmap_taskset_sscanf (hwloc_bitmap_t *bitmap,* const char * __hwloc_restrict *string*)

Parse a taskset-specific bitmap string and stores it in bitmap `bitmap`.

15.28.4.41 HWLOC_DECLSPEC 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.

15.28.4.42 HWLOC_DECLSPEC unsigned long hwloc_bitmap_to_ulong (hwloc_const_bitmap_t *bitmap*)

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

15.28.4.43 HWLOC_DECLSPEC 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.

15.28.4.44 HWLOC_DECLSPEC 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`.

15.28.4.45 HWLOC_DECLSPEC void hwloc_bitmap_zero (hwloc_bitmap_t *bitmap*)

Empty the bitmap `bitmap`.

15.29 Helpers for manipulating glibc sched affinity

Functions

- static __hwloc_inline int hwloc_cpuset_to_glibc_sched_affinity (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_const_cpuset_t hwlocset, cpu_set_t *schedset, size_t schedsetsize)
- static __hwloc_inline int hwloc_cpuset_from_glibc_sched_affinity (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_cpuset_t hwlocset, const cpu_set_t *schedset, size_t schedsetsize)

15.29.1 Detailed Description

15.29.2 Function Documentation

15.29.2.1 static __hwloc_inline int hwloc_cpuset_from_glibc_sched_affinity (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_cpuset_t hwlocset, const cpu_set_t * schedset, size_t schedsetsize) [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

15.29.2.2 static __hwloc_inline int hwloc_cpuset_to_glibc_sched_affinity (hwloc_topology_t topology __hwloc_attribute_unused, hwloc_const_cpuset_t hwlocset, cpu_set_t * schedset, size_t schedsetsize) [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

15.30 Linux-only helpers

Functions

- HWLOC_DECLSPEC int hwloc_linux_parse_cpumap_file (FILE *file, hwloc_cpuset_t set)
- HWLOC_DECLSPEC int hwloc_linux_set_tid_cpubind (hwloc_topology_t topology, pid_t tid, hwloc_const_cpuset_t set)
- HWLOC_DECLSPEC int hwloc_linux_get_tid_cpubind (hwloc_topology_t topology, pid_t tid, hwloc_cpuset_t set)

15.30.1 Detailed Description

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

15.30.2 Function Documentation

15.30.2.1 HWLOC_DECLSPEC 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.

15.30.2.2 HWLOC_DECLSPEC 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.

15.30.2.3 HWLOC_DECLSPEC 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.

15.31 Helpers for manipulating Linux libnuma unsigned long masks

Functions

- static __hwloc_inline int hwloc_cpuset_to_linux_libnuma_ulongs (hwloc_topology_t topology, hwloc_const_cpuset_t cpuset, unsigned long *mask, unsigned long *maxnode)
- static __hwloc_inline int hwloc_nodeset_to_linux_libnuma_ulongs (hwloc_topology_t topology, hwloc_const_nodeset_t nodeset, unsigned long *mask, unsigned long *maxnode)
- static __hwloc_inline int hwloc_cpuset_from_linux_libnuma_ulongs (hwloc_topology_t topology, hwloc_cpuset_t cpuset, const unsigned long *mask, unsigned long maxnode)
- static __hwloc_inline int hwloc_nodeset_from_linux_libnuma_ulongs (hwloc_topology_t topology, hwloc_nodeset_t nodeset, const unsigned long *mask, unsigned long maxnode)

15.31.1 Detailed Description

15.31.2 Function Documentation

15.31.2.1 static __hwloc_inline int hwloc_cpuset_from_linux_libnuma_ulongs (hwloc_topology_t *topology*, hwloc_cpuset_t *cpuset*, const unsigned long * *mask*, unsigned long *maxnode*) [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).

15.31.2.2 static __hwloc_inline int hwloc_cpuset_to_linux_libnuma_ulongs (hwloc_topology_t *topology*, hwloc_const_cpuset_t *cpuset*, unsigned long * *mask*, unsigned long * *maxnode*) [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.

15.31.2.3 static __hwloc_inline int hwloc_nodeset_from_linux_libnuma_ulongs (hwloc_topology_t *topology*, hwloc_nodeset_t *nodeset*, const unsigned long * *mask*, unsigned long *maxnode*) [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).

15.31.2.4 static __hwloc_inline int hwloc_nodeset_to_linux_libnuma_ulongs (hwloc_topology_t *topology*, hwloc_const_nodeset_t *nodeset*, unsigned long * *mask*, unsigned long * *maxnode*) [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.

15.32 Helpers for manipulating Linux libnuma bitmask

Functions

- static __hwloc_inline struct bitmask * hwloc_cpuset_to_linux_libnuma_bitmask (hwloc_topology_t topology, hwloc_const_cpuset_t cpuset) __hwloc_attribute_malloc
- static __hwloc_inline struct bitmask * hwloc_nodeset_to_linux_libnuma_bitmask (hwloc_topology_t topology, hwloc_const_nodeset_t nodeset) __hwloc_attribute_malloc
- static __hwloc_inline int hwloc_cpuset_from_linux_libnuma_bitmask (hwloc_topology_t topology, hwloc_cpuset_t cpuset, const struct bitmask *bitmask)
- static __hwloc_inline int hwloc_nodeset_from_linux_libnuma_bitmask (hwloc_topology_t topology, hwloc_nodeset_t nodeset, const struct bitmask *bitmask)

15.32.1 Detailed Description

15.32.2 Function Documentation

15.32.2.1 static __hwloc_inline int hwloc_cpuset_from_linux_libnuma_bitmask (hwloc_topology_t *topology,* hwloc_cpuset_t *cpuset,* const struct bitmask * *bitmask*) [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.

15.32.2.2 static __hwloc_inline struct bitmask * hwloc_cpuset_to_linux_libnuma_bitmask (hwloc_topology_t *topology,* hwloc_const_cpuset_t *cpuset*) [static], [read]

Convert hwloc CPU set `cpuset` into the returned libnuma bitmask.

The returned bitmask should later be freed with numa_bitmask_free.

This function may be used before calling many numa_ functions that use a struct bitmask as an input parameter.

Returns

newly allocated struct bitmask.

15.32.2.3 static __hwloc_inline int hwloc_nodeset_from_linux_libnuma_bitmask (hwloc_topology_t *topology,* hwloc_nodeset_t *nodeset,* const struct bitmask * *bitmask*) [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.

15.32.2.4 static __hwloc_inline struct bitmask * hwloc_nodeset_to_linux_libnuma_bitmask (hwloc_topology_t *topology,* hwloc_const_nodeset_t *nodeset*) [static], [read]

Convert hwloc NUMA node set `nodeset` into the returned libnuma bitmask.

The returned bitmask should later be freed with numa_bitmask_free.

This function may be used before calling many numa_ functions that use a struct bitmask as an input parameter.

Returns

newly allocated struct bitmask.

15.33 Helpers for manipulating Linux libnuma nodemask_t

Functions

- static __hwloc_inline int hwloc_cpuset_to_linux_libnuma_nodemask (hwloc_topology_t topology, hwloc_const_cpuset_t cpuset, nodemask_t *nodemask)
- static __hwloc_inline int hwloc_nodeset_to_linux_libnuma_nodemask (hwloc_topology_t topology, hwloc_const_nodeset_t nodeset, nodemask_t *nodemask)
- static __hwloc_inline int hwloc_cpuset_from_linux_libnuma_nodemask (hwloc_topology_t topology, hwloc_cpuset_t cpuset, const nodemask_t *nodemask)
- static __hwloc_inline int hwloc_nodeset_from_linux_libnuma_nodemask (hwloc_topology_t topology, hwloc_nodeset_t nodeset, const nodemask_t *nodemask)

15.33.1 Detailed Description

Note

The Linux libnuma nodemask_t interface is deprecated and its implementation is at least incorrect with respect to sparse NUMA node ids. It is strongly advised to use struct bitmask instead of nodemask_t, or even to use hwloc directly.

15.33.2 Function Documentation

15.33.2.1 static __hwloc_inline int hwloc_cpuset_from_linux_libnuma_nodemask (hwloc_topology_t *topology*, hwloc_cpuset_t *cpuset*, const nodemask_t * *nodemask*) [static]

Convert libnuma nodemask `nodemask` into hwloc CPU set `cpuset`.

This function may be used before calling some old libnuma functions that use a nodemask_t as an output parameter.

15.33.2.2 static __hwloc_inline int hwloc_cpuset_to_linux_libnuma_nodemask (hwloc_topology_t *topology*, hwloc_const_cpuset_t *cpuset*, nodemask_t * *nodemask*) [static]

Convert hwloc CPU set `cpuset` into libnuma nodemask `nodemask`.

This function may be used before calling some old libnuma functions that use a nodemask_t as an input parameter.

15.33.2.3 static __hwloc_inline int hwloc_nodeset_from_linux_libnuma_nodemask (hwloc_topology_t *topology*, hwloc_nodeset_t *nodeset*, const nodemask_t * *nodemask*) [static]

Convert libnuma nodemask `nodemask` into hwloc NUMA node set `nodeset`.

This function may be used before calling some old libnuma functions that use a nodemask_t as an output parameter.

15.33.2.4 static __hwloc_inline int hwloc_nodeset_to_linux_libnuma_nodemask (hwloc_topology_t *topology*, hwloc_const_nodeset_t *nodeset*, nodemask_t * *nodemask*) [static]

Convert hwloc NUMA node set `nodeset` into libnuma nodemask `nodemask`.

This function may be used before calling some old libnuma functions that use a nodemask_t as an input parameter.

15.34 CUDA Driver API Specific Functions

Functions

- static __hwloc_inline int hwloc_cuda_get_device_pci_ids (hwloc_topology_t topology __hwloc_attribute_unused, CUdevice cudevice, int *domain, int *bus, int *dev)
- static __hwloc_inline int hwloc_cuda_get_device_cpuset (hwloc_topology_t topology __hwloc_attribute_unused, CUdevice cudevice, hwloc_cpuset_t set)
- static __hwloc_inline hwloc_obj_t hwloc_cuda_get_device_pcidev (hwloc_topology_t topology, CUdevice cudevice)

15.34.1 Detailed Description

15.34.2 Function Documentation

15.34.2.1 static __hwloc_inline int hwloc_cuda_get_device_cpuset (hwloc_topology_t topology __hwloc_attribute_unused, CUdevice *cudevice*, hwloc_cpuset_t *set*) [static]

Get the CPU set of logical processors that are physically close to device `cudevice`.

For the given CUDA Driver API device `cudevice`, read the corresponding kernel-provided cpumap file and return the corresponding CPU set. This function is currently only implemented in a meaningful way for Linux; other systems will simply get a full cpuset.

Topology `topology` must match the current machine.

15.34.2.2 static __hwloc_inline int hwloc_cuda_get_device_pci_ids (hwloc_topology_t topology __hwloc_attribute_unused, CUdevice *cudevice*, int * *domain*, int * *bus*, int * *dev*) [static]

Return the domain, bus and device IDs of device `cudevice`.

15.34.2.3 static __hwloc_inline hwloc_obj_t hwloc_cuda_get_device_pcidev (hwloc_topology_t *topology*, CUdevice *cudevice*) [static]

Get the hwloc object for the PCI device corresponding to device `cudevice`.

For the given CUDA Runtime API device `cudevice`, return the hwloc PCI object containing the device. Returns NULL if there is none.

IO devices detection must be enabled in topology `topology`.

15.35 CUDA Runtime API Specific Functions

Functions

- static __hwloc_inline int hwloc_cudart_get_device_pci_ids (hwloc_topology_t topology __hwloc_attribute_unused, int device, int *domain, int *bus, int *dev)
- static __hwloc_inline int hwloc_cudart_get_device_cpuset (hwloc_topology_t topology __hwloc_attribute_unused, int device, hwloc_cpuset_t set)
- static __hwloc_inline hwloc_obj_t hwloc_cudart_get_device_pcidev (hwloc_topology_t topology, int device)

15.35.1 Detailed Description

15.35.2 Function Documentation

15.35.2.1 static __hwloc_inline int hwloc_cudart_get_device_cpuset (hwloc_topology_t topology __hwloc_attribute_unused, int device, hwloc_cpuset_t set) [static]

Get the CPU set of logical processors that are physically close to device `device`.

For the given CUDA Runtime API device `device`, read the corresponding kernel-provided cpumap file and return the corresponding CPU set. This function is currently only implemented in a meaningful way for Linux; other systems will simply get a full cpuset.

Topology `topology` must match the current machine.

15.35.2.2 static __hwloc_inline int hwloc_cudart_get_device_pci_ids (hwloc_topology_t topology __hwloc_attribute_unused, int device, int * domain, int * bus, int * dev) [static]

Return the domain, bus and device IDs of device `device`.

15.35.2.3 static __hwloc_inline hwloc_obj_t hwloc_cudart_get_device_pcidev (hwloc_topology_t topology, int device) [static]

Get the hwloc object for the PCI device corresponding to device `device`.

For the given CUDA Runtime API device `device`, return the hwloc PCI object containing the device. Returns NULL if there is none.

IO devices detection must be enabled in topology `topology`.

15.36 OpenFabrics-Specific Functions

Functions

- static __hwloc_inline int hwloc_ibv_get_device_cpuset (hwloc_topology_t topology __hwloc_attribute_unused, struct ibv_device *ibdev, hwloc_cpuset_t set)

15.36.1 Detailed Description

15.36.2 Function Documentation

15.36.2.1 static __hwloc_inline int hwloc_ibv_get_device_cpuset (hwloc_topology_t topology __hwloc_attribute_unused, struct ibv_device * ibdev, hwloc_cpuset_t set) [static]

Get the CPU set of logical processors that are physically close to device `ibdev`.

For the given OpenFabrics device `ibdev`, read the corresponding kernel-provided cpumap file and return the corresponding CPU set. This function is currently only implemented in a meaningful way for Linux; other systems will simply get a full cpuset.

Topology `topology` must match the current machine.

15.37 Myrinet Express-Specific Functions

Functions

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

15.37.1 Detailed Description

15.37.2 Function Documentation

15.37.2.1 static __hwloc_inline int hwloc_mx_board_get_device_cpuset (hwloc_topology_t *topology,* unsigned *id,* hwloc_cpuset_t *set*) [static]

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

For the given Myrinet Express board index `id`, read the OS-provided NUMA node and return the corresponding CPU set.

Topology `topology` must match the current machine.

15.37.2.2 static __hwloc_inline int hwloc_mx_endpoint_get_device_cpuset (hwloc_topology_t *topology,* mx_endpoint_t *endpoint,* hwloc_cpuset_t *set*) [static]

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

For the given Myrinet Express endpoint `endpoint`, read the OS-provided NUMA node and return the corresponding CPU set.

Topology `topology` must match the current machine.

Chapter 16

Data Structure Documentation

16.1 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

16.1.1 Detailed Description

Bridge specific Object Attribues.

16.1.2 Field Documentation

16.1.2.1 unsigned hwloc_obj_attr_u::hwloc_bridge_attr_s::depth

16.1.2.2 unsigned short hwloc_obj_attr_u::hwloc_bridge_attr_s::domain

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

16.1.2.4 hwloc_obj_bridge_type_t hwloc_obj_attr_u::hwloc_bridge_attr_s::downstream_type

16.1.2.5 struct hwloc_pcidev_attr_s hwloc_obj_attr_u::hwloc_bridge_attr_s::pci

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

16.1.2.7 unsigned char hwloc_obj_attr_u::hwloc_bridge_attr_s::secondary_bus

16.1.2.8 unsigned char hwloc_obj_attr_u::hwloc_bridge_attr_s::subordinate_bus

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

16.1.2.10 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

16.2 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

16.2.1 Detailed Description

Cache-specific Object Attributes.

16.2.2 Field Documentation

16.2.2.1 int hwloc_obj_attr_u::hwloc_cache_attr_s::associativity

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

16.2.2.2 unsigned hwloc_obj_attr_u::hwloc_cache_attr_s::depth

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

16.2.2.3 unsigned hwloc_obj_attr_u::hwloc_cache_attr_s::linesize

Cache-line size in bytes.

16.2.2.4 hwloc_uint64_t hwloc_obj_attr_u::hwloc_cache_attr_s::size

Size of cache in bytes.

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

- hwloc.h

16.3 hwloc_distances_s Struct Reference

`#include <hwloc.h>`

Data Fields

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

16.3.1 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), as defined by the ACPI SLIT specification.

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

16.3.2 Field Documentation

16.3.2.1 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. 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.

16.3.2.2 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.

16.3.2.3 float hwloc_distances_s::latency_max

The maximal value in the latency matrix.

16.3.2.4 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.

16.3.2.5 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

16.4 hwloc_obj_attr_u::hwloc_group_attr_s Struct Reference

`#include <hwloc.h>`

Data Fields

- unsigned depth

16.4.1 Detailed Description

Group-specific Object Attributes.

16.4.2 Field Documentation

16.4.2.1 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

16.5 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

16.5.1 Detailed Description

Structure of a topology object.

Applications must not modify any field except hwloc_obj.userdata.

16.5.2 Field Documentation

16.5.2.1 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.

16.5.2.2 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.

16.5.2.3 unsigned hwloc_obj::arity

Number of children.

16.5.2.4 union hwloc_obj_attr_u* hwloc_obj::attr

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

16.5.2.5 struct hwloc_obj** hwloc_obj::children

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

16.5.2.6 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.

16.5.2.7 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.

16.5.2.8 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.

16.5.2.9 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.

16.5.2.10 struct hwloc_distances_s** hwloc_obj::distances

Distances between all objects at same depth below this object.

16.5.2.11 unsigned hwloc_obj::distances_count

16.5.2.12 struct hwloc_obj* hwloc_obj::first_child

First child.

16.5.2.13 struct hwloc_obj_info_s* hwloc_obj::infos

Array of stringified info type=name.

16.5.2.14 unsigned hwloc_obj::infos_count

Size of infos array.

16.5.2.15 struct hwloc_obj* hwloc_obj::last_child

Last child.

16.5.2.16 unsigned hwloc_obj::logical_index

Horizontal index in the whole list of similar objects, could be a "cousin_rank" since it's the rank within the "cousin" list below.

16.5.2.17 struct hwloc_obj_memory_s hwloc_obj::memory

Memory attributes.

16.5.2.18 char* hwloc_obj::name

Object description if any.

16.5.2.19 struct hwloc_obj* hwloc_obj::next_cousin

Next object of same type and depth.

16.5.2.20 struct hwloc_obj* hwloc_obj::next_sibling

Next object below the same parent.

16.5.2.21 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.

16.5.2.22 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.

16.5.2.23 unsigned hwloc_obj::os_index

OS-provided physical index number.

16.5.2.24 signed hwloc_obj::os_level

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

16.5.2.25 struct hwloc_obj* hwloc_obj::parent

Parent, NULL if root (system object)

16.5.2.26 struct hwloc_obj* hwloc_obj::prev_cousin

Previous object of same type and depth.

16.5.2.27 struct hwloc_obj* hwloc_obj::prev_sibling

Previous object below the same parent.

16.5.2.28 unsigned hwloc_obj::sibling_rank

Index in parent's children[] array.

16.5.2.29 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.

16.5.2.30 hwloc_obj_type_t hwloc_obj::type

Type of object.

16.5.2.31 void* hwloc_obj::userdata

Application-given private data pointer, initialized to NULL, use it as you wish.

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

- hwloc.h

16.6 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

16.6.1 Detailed Description

Object type-specific Attributes.

16.6.2 Field Documentation

16.6.2.1 struct hwloc_obj_attr_u::hwloc_bridge_attr_s hwloc_obj_attr_u::bridge

16.6.2.2 struct hwloc_obj_attr_u::hwloc_cache_attr_s hwloc_obj_attr_u::cache

16.6.2.3 struct hwloc_obj_attr_u::hwloc_group_attr_s hwloc_obj_attr_u::group

16.6.2.4 struct hwloc_obj_attr_u::hwloc_osdev_attr_s hwloc_obj_attr_u::osdev

16.6.2.5 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

16.7 hwloc_obj_info_s Struct Reference

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

Data Fields

- char * name
- char * value

16.7.1 Detailed Description

Object info.

16.7.2 Field Documentation

16.7.2.1 char* hwloc_obj_info_s::name

Info name.

16.7.2.2 char* hwloc_obj_info_s::value

Info value.

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

- hwloc.h

16.8 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

16.8.1 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.

16.8.2 Field Documentation

16.8.2.1 hwloc_uint64_t hwloc_obj_memory_s::hwloc_obj_memory_page_type_s::count

Number of pages of this size.

16.8.2.2 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

16.9 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

16.9.1 Detailed Description

Object memory.

16.9.2 Field Documentation

16.9.2.1 hwloc_uint64_t hwloc_obj_memory_s::local_memory

Local memory (in bytes)

16.9.2.2 struct hwloc_obj_memory_s::hwloc_obj_memory_page_type_s * hwloc_obj_memory_s::page_types

16.9.2.3 unsigned hwloc_obj_memory_s::page_types_len

Size of array `page_types`.

16.9.2.4 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

16.10 hwloc_obj_attr_u::hwloc_osdev_attr_s Struct Reference

`#include <hwloc.h>`

Data Fields

- hwloc_obj_osdev_type_t type

16.10.1 Detailed Description

OS Device specific Object Attributes.

16.10.2 Field Documentation

16.10.2.1 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

16.11 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

16.11.1 Detailed Description

PCI Device specific Object Attributes.

16.11.2 Field Documentation

16.11.2.1 unsigned char hwloc_obj_attr_u::hwloc_pcidev_attr_s::bus

16.11.2.2 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::class_id

16.11.2.3 unsigned char hwloc_obj_attr_u::hwloc_pcidev_attr_s::dev

16.11.2.4 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::device_id

16.11.2.5 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::domain

16.11.2.6 unsigned char hwloc_obj_attr_u::hwloc_pcidev_attr_s::func

16.11.2.7 float hwloc_obj_attr_u::hwloc_pcidev_attr_s::linkspeed

16.11.2.8 unsigned char hwloc_obj_attr_u::hwloc_pcidev_attr_s::revision

16.11.2.9 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::subdevice_id

16.11.2.10 unsigned short hwloc_obj_attr_u::hwloc_pcidev_attr_s::subvendor_id

16.11.2.11 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

16.12 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

16.12.1 Detailed Description

Flags describing actual PU binding support for this topology.

16.12.2 Field Documentation

16.12.2.1 unsigned char hwloc_topology_cpubind_support::get_proc_cpubind

Getting the binding of a whole given process is supported.

16.12.2.2 unsigned char hwloc_topology_cpubind_support::get_proc_last_cpu_location

Getting the last processors where a whole process ran is supported

16.12.2.3 unsigned char hwloc_topology_cpubind_support::get_thisproc_cpubind

Getting the binding of the whole current process is supported.

16.12.2.4 unsigned char hwloc_topology_cpubind_support::get_thisproc_last_cpu_location

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

16.12.2.5 unsigned char hwloc_topology_cpubind_support::get_thisthread_cpubind

Getting the binding of the current thread only is supported.

16.12.2.6 unsigned char hwloc_topology_cpubind_support::get_thisthread_last_cpu_location

Getting the last processors where the current thread ran is supported

16.12.2.7 unsigned char hwloc_topology_cpubind_support::get_thread_cpubind

Getting the binding of a given thread only is supported.

16.12.2.8 unsigned char hwloc_topology_cpubind_support::set_proc_cpubind

Binding a whole given process is supported.

16.12.2.9 unsigned char hwloc_topology_cpubind_support::set_thisproc_cpubind

Binding the whole current process is supported.

16.12.2.10 unsigned char hwloc_topology_cpubind_support::set_thisthread_cpubind

Binding the current thread only is supported.

16.12.2.11 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

16.13 hwloc_topology_discovery_support Struct Reference

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

Data Fields

- unsigned char pu

16.13.1 Detailed Description

Flags describing actual discovery support for this topology.

16.13.2 Field Documentation

16.13.2.1 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

16.14 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

16.14.1 Detailed Description

Flags describing actual memory binding support for this topology.

16.14.2 Field Documentation

16.14.2.1 unsigned char hwloc_topology_membind_support::alloc_membind

Allocating a bound memory area is supported.

16.14.2.2 unsigned char hwloc_topology_membind_support::bind_membind

Bind policy is supported.

16.14.2.3 unsigned char hwloc_topology_membind_support::firsttouch_membind

First-touch policy is supported.

16.14.2.4 unsigned char hwloc_topology_membind_support::get_area_membind

Getting the binding of a given memory area is supported.

16.14.2.5 unsigned char hwloc_topology_membind_support::get_proc_membind

Getting the binding of a whole given process is supported.

16.14.2.6 unsigned char hwloc_topology_membind_support::get_thisproc_membind

Getting the binding of the whole current process is supported.

16.14.2.7 unsigned char hwloc_topology_membind_support::get_thisthread_membind

Getting the binding of the current thread only is supported.

16.14.2.8 unsigned char hwloc_topology_membind_support::interleave_membind

Interleave policy is supported.

16.14.2.9 unsigned char hwloc_topology_membind_support::migrate_membind

Migration flags is supported.

16.14.2.10 unsigned char hwloc_topology_membind_support::nexttouch_membind

Next-touch migration policy is supported.

16.14.2.11 unsigned char hwloc_topology_membind_support::replicate_membind

Replication policy is supported.

16.14.2.12 unsigned char hwloc_topology_membind_support::set_area_membind

Binding a given memory area is supported.

16.14.2.13 unsigned char hwloc_topology_membind_support::set_proc_membind

Binding a whole given process is supported.

16.14.2.14 unsigned char hwloc_topology_membind_support::set_thisproc_membind

Binding the whole current process is supported.

16.14.2.15 unsigned char hwloc_topology_membind_support::set_thisthread_membind

Binding the current thread only is supported.

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

- hwloc.h

16.15 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

16.15.1 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.

16.15.2 Field Documentation

16.15.2.1 struct hwloc_topology_cpubind_support* hwloc_topology_support::cpubind

16.15.2.2 struct hwloc_topology_discovery_support* hwloc_topology_support::discovery

16.15.2.3 struct hwloc_topology_membind_support* hwloc_topology_support::membind

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

- hwloc.h

Index