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, and processing units (logical processors or "threads"). hwloc also gathers various attributes such as cache and memory information, and is portable across a variety of different operating systems and platforms.

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

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 is 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. More new features and more PLPA-like features will be added to hwloc over time. See Switching from PLPA to hwloc for more details about converting your application from PLPA 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

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.

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.

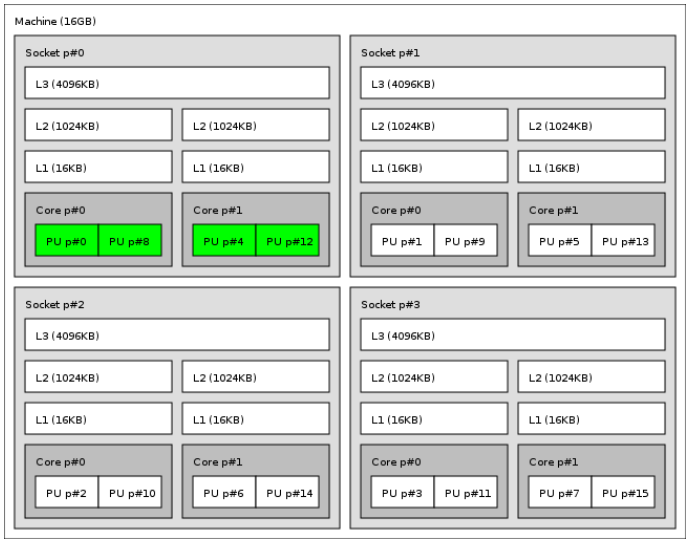
Installation by itself is the fairly common GNU-based process:

```
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. Similarly, lstopo's XML support requires the libxml2 development package.

1.3 CLI Examples

On a 4-socket 2-core machine with hyperthreading, the `lstopo` tool may show the following graphic output:



Here's the equivalent output in textual form:

```
Machine (16GB)
  Socket #0 + L3 #0 (4096KB)
    L2 #0 (1024KB) + L1 #0 (16KB) + Core #0
      PU #0 (phys=0)
      PU #1 (phys=8)
    L2 #1 (1024KB) + L1 #1 (16KB) + Core #1
      PU #2 (phys=4)
```

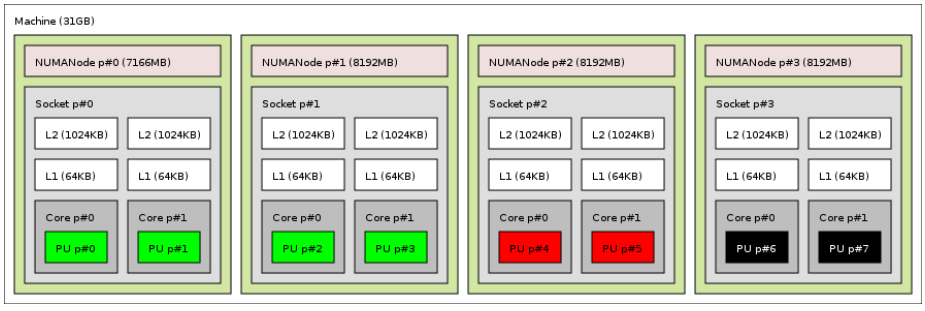
```
    PU #3 (phys=12)
  Socket #1 + L3 #1 (4096KB)
    L2 #2 (1024KB) + L1 #2 (16KB) + Core #2
      PU #4 (phys=1)
      PU #5 (phys=9)
    L2 #3 (1024KB) + L1 #3 (16KB) + Core #3
      PU #6 (phys=5)
      PU #7 (phys=13)
  Socket #2 + L3 #2 (4096KB)
    L2 #4 (1024KB) + L1 #4 (16KB) + Core #4
      PU #8 (phys=2)
      PU #9 (phys=10)
    L2 #5 (1024KB) + L1 #5 (16KB) + Core #5
      PU #10 (phys=6)
      PU #11 (phys=14)
  Socket #3 + L3 #3 (4096KB)
    L2 #6 (1024KB) + L1 #6 (16KB) + Core #6
      PU #12 (phys=3)
      PU #13 (phys=11)
    L2 #7 (1024KB) + L1 #7 (16KB) + Core #7
      PU #14 (phys=7)
      PU #15 (phys=15)
```

Finally, here's the equivalent output in XML (only the first socket 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 graphic output:



Here's the equivalent output in textual form:

```
Machine (64GB)
  NUMANode #0 (phys=0 8190MB) + Socket #0
    L2 #0 (1024KB) + L1 #0 (64KB) + Core #0 + PU #0 (phys=0)
    L2 #1 (1024KB) + L1 #1 (64KB) + Core #1 + PU #1 (phys=1)
```