Hardware locality (hwloc)
Managing hardware affinities for HPC applications
Machines are getting increasingly complex
Machines are getting increasingly complex

- Multiple processors, manycores, SMT, …
Machines are getting increasingly complex

- Multiple processors, manycores, SMT, …
- Shared caches between some cores
Machines are getting increasingly complex

- Multiple processors, manycores, SMT, …
- Shared caches between some cores
- Multiple memory nodes (NUMA)
Machines are getting increasingly complex

- Multiple processors, manycores, SMT
- Shared caches between some cores
- Multiple memory nodes (NUMA)
- NICs, GPUs, …
Machines are getting increasingly complex

- Multiple processors, manycores, SMT, ...
- Shared caches between some cores
- Multiple memory nodes (NUMA)
- NICs, GPUs, ...

![Diagram of NUMAnode P#0 (24GB)]

- Socket P#0
  - L3 (12MB)
  - L2 (256KB)
  - L1 (32KB)
  - Core P#0: PU P#0, PU P#12
  - Core P#1: PU P#1, PU P#13
  - Core P#2: PU P#2, PU P#14
  - Core P#8: PU P#3, PU P#15
  - Core P#9: PU P#4, PU P#16
  - Core P#10: PU P#5, PU P#17

![Diagram of NUMAnode P#1 (24GB)]

- Socket P#1
  - L3 (12MB)
  - L2 (256KB)
  - L1 (32KB)
  - Core P#0: PU P#6, PU P#18
  - Core P#1: PU P#7, PU P#19
  - Core P#2: PU P#8, PU P#20
  - Core P#8: PU P#9, PU P#21
  - Core P#9: PU P#10, PU P#22
  - Core P#10: PU P#11, PU P#23
Affinities are one of the key performance criteria

Dilemma
• Use cores 0 & 1 to share cache and improve synchronization cost?
• Use cores 0 & 2 to maximize memory bandwidth?
• How to choose portably?

Depends both on the application structure and the machine structure
What’s in my machine?

Machine (2048MB)

NUMANode P#0 (1024MB)

Socket P#0
- Core P#0
  - PU P#0
  - PU P#1
- Core P#1
  - PU P#2
  - PU P#3
- Core P#2
  - PU P#4
  - PU P#5

Socket P#1
- Core P#6
  - PU P#12
  - PU P#13
- Core P#7
  - PU P#14
  - PU P#15
- Core P#8
  - PU P#16
  - PU P#17
- Core P#9
  - PU P#18
  - PU P#19
- Core P#10
  - PU P#20
  - PU P#21
- Core P#11
  - PU P#22
  - PU P#23
Or maybe it’s a bit different

Machine (2048MB)

NUMANode P#0 (1024MB)

Socket P#0
Core P#0
PU P#0
PU P#12
Core P#1
PU P#1
PU P#13
Core P#2
PU P#2
PU P#14
Core P#3
PU P#3
PU P#15
Core P#4
PU P#4
PU P#16
Core P#5
PU P#5
PU P#17

NUMANode P#1 (1024MB)

Socket P#1
Core P#6
PU P#6
PU P#18
Core P#7
PU P#7
PU P#19
Core P#8
PU P#8
PU P#20
Core P#9
PU P#9
PU P#21
Core P#10
PU P#10
PU P#22
Core P#11
PU P#11
PU P#23
Wait!? After rebooting, it’s different again…
Hardware organization is unpredictable

You may know what you bought…

… but you can’t assume how processors, cores, threads, … will be \textit{physically} numbered

- Depends on vendor
- May change after BIOS or OS upgrade
Gathering topology information and binding threads/memory is difficult

Lack of generic, uniform interface

- OS specific
  - /proc, /sys, rset, sysctl, lgrp, kstat, CPUID, ...
  - setaffinity, rset, ldom_bind, radset, affinity_set, ...
  - mbind, rset, mmap, nmadvise, affinity_set, ...

- Distribution specific
- Low-level APIs

Evolving technology

- E.g. : AMD Bulldozer “half-cores”, Intel SCC “tiles”
Gathering topology information and binding threads/memory is difficult

Lack of generic, uniform interface

- OS specific
  - /proc, /sys, rset, sysctl, lgrp, kstat, CPUID, ...
  - setaffinity, rset, ldom_bind, radset, affinity_set, ...
  - mbind, rset, mmap, nmadvise, affinity_set, ...

- Distribution specific

- Low-level APIs

Evolving technology

- E.g. : AMD Bulldozer “half-cores”, Intel SCC “tiles”

⇒ Need generic tools & abstract API

- Logical resource identification
Hwloc
Portable Hardware Locality

- Portable topology information
- Portable binding toolset
Hwloc

• Joint development
  - Runtime group + Open-MPI/Cisco
  - Libtopology (initially part of the Marcel scheduler)
  - PLPA

• Two parts
  - Set of command line tools (lstopo, hwloc-bind, calc, etc.)
  - C API + library, Perl and Python bindings

• Portable: Linux, Solaris, AIX, HP-UX, FreeBSD, Darwin, Windows

• BSD-3 license

• Used by a lot of projects: most MPI, runtimes, batch scheds, ...

http://www.open-mpi.org/projects/hwloc/
Hwloc's view of the hardware

Tree of objects

- Machines, memory nodes, sockets, caches, cores, threads, …
  - Logically ordered
- Grouping of similar objects based on distances between them
- Many attributes
  - Memory node size
  - Cache type, size and line size
  - Machine model
  - Physical ordering
Tools
- Nice output
- shell-prone utilities
Istopo – Displaying topology information

Textual rendering: Istopo -

Machine (total=1024MB)
  NUMANode #0 (phys=0 local=1024MB)
    Socket #0 (phys=0)
      L3Cache #0 (16MB)
      L2Cache #0 (4096KB)
      L1Cache #0 (32KB)
      Core #0 (phys=0)
        PU #0 (phys=0)
        PU #1 (phys=2)
      L2Cache #1 (4096KB)
      L1Cache #1 (32KB)
      Core #1 (phys=2)
        PU #2 (phys=1)
        PU #3 (phys=3)

Graphical rendering
Istopo – Displaying topology information

- Lstopo supports various output formats
  - .fig, .pdf, .ps, .png, .svg

$ lstopo output.png

- It also supports XML format
  - Permits to save and quickly restore instead of re-performing detection
  - Permits to store other machine's topology for reference
Istopo – Displaying topology information

Even text-mode pseudo-graphical display!  
Istopo -.txt
Istopo – Displaying topology information

Various output options, useful for slides :)

• --horiz: force horizontal layout
• --ignore cache: drop caches from the output
• --restrict <cpuset>: restrict output to a mask of processors
• ...

Istopo – Displaying topology information

Synthetic topology, useful for slides too :)
$ lstopo --input "node:1 socket:1 cache:1 cache:2 cache:1 core:1 pu:2"

… and a lot more, see lstopo --help
hwloc-distances – show object distances

Notably NUMA distances:

$ ./utils/hwloc-distances

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.000</td>
<td>1.600</td>
<td>1.600</td>
<td>2.200</td>
<td>1.600</td>
<td>2.200</td>
<td>1.600</td>
<td>2.200</td>
</tr>
<tr>
<td>1</td>
<td>1.600</td>
<td>1.000</td>
<td>2.200</td>
<td>1.600</td>
<td>2.200</td>
<td>1.600</td>
<td>1.600</td>
<td>2.200</td>
</tr>
<tr>
<td>2</td>
<td>1.600</td>
<td>2.200</td>
<td>1.000</td>
<td>1.600</td>
<td>1.600</td>
<td>2.200</td>
<td>1.600</td>
<td>2.200</td>
</tr>
<tr>
<td>3</td>
<td>2.200</td>
<td>1.600</td>
<td>1.600</td>
<td>1.000</td>
<td>1.600</td>
<td>2.200</td>
<td>2.200</td>
<td>1.600</td>
</tr>
<tr>
<td>4</td>
<td>1.600</td>
<td>2.200</td>
<td>1.600</td>
<td>1.600</td>
<td>1.000</td>
<td>1.600</td>
<td>1.600</td>
<td>1.600</td>
</tr>
<tr>
<td>5</td>
<td>2.200</td>
<td>1.600</td>
<td>2.200</td>
<td>2.200</td>
<td>1.600</td>
<td>1.000</td>
<td>1.600</td>
<td>1.600</td>
</tr>
<tr>
<td>6</td>
<td>1.600</td>
<td>1.600</td>
<td>1.600</td>
<td>2.200</td>
<td>1.600</td>
<td>1.600</td>
<td>1.000</td>
<td>1.600</td>
</tr>
<tr>
<td>7</td>
<td>2.200</td>
<td>2.200</td>
<td>2.200</td>
<td>1.600</td>
<td>1.600</td>
<td>1.600</td>
<td>1.600</td>
<td>1.000</td>
</tr>
</tbody>
</table>
Physical indexes

As returned by the OS, default lstopo output
As mentioned earlier: often rather odd, depends on motherboard, BIOS, moon, ...
Logical indexes

As computed by hwloc, lstopo -l
Always represents proximity (depth-first walk)
Sets of CPUs

Hwloc tools have several ways to designate a set of cpus

- A set of objects:
  socket:0 core:4-7

- Can be more specific: two first cores of second socket:
  socket:1.core:0-1

- A bitmask:
  0x44

- CPUs close to a given PCI device
  pci=01:00.0

- Or to an OS device
  os=eth0
hwloc-calc - compute CPU sets

Permits to convert between ways to designate CPU sets, and make combinations:

$ hwloc-calc socket:1
0x000000f0

$ hwloc-calc os=eth0
0x00005555

$ hwloc-calc socket:2 ~PU:even
0x00000c00

$ hwloc-calc --number-of core socket:1
4

$ hwloc-calc --intersect PU socket:1
4,5,6,7
hwloc-bind – bind process

Bind a new process to a given set of CPUs:
$ hwloc-bind socket:1 -- mycommand

Bind an existing process:
$ hwloc-bind --pid 1234 socket:1

Bind memory:
$ hwloc-bind --membind node:1 --cpubind node:1.socket:0 -- mycommand

Distribute memory:
$ hwloc-bind --membind --mempolicy interleave all -- mycommand
Istopo – show bound processes

$ lstopo –ps

Also hwloc-ps:
6945 core:0  sh
hwloc-assembler – combine trees

Permits to create network topologies
$ hwloc-assembler combined.xml machine1.xml machine2.xml
$ lstopo --input combined.xml
Hands-on: 1st part
http://runtime.bordeaux.inria.fr/hwloc/hwloc_tutorial.html
Programming Interface
- browsing objects
- CPU/node set operations
- CPU/memory binding
Initialization / termination

Should be trivial enough :)

```c
hwloc_topology_t t;

hwloc_topology_init(&t); // initialization
Optional detection configuration...
hwloc_topology_load(t); // actual detection

Play with it...
nbcores = hwloc_get_nbobjs_by_type(t, HWLOC_OBJ_CORE);

hwloc_topology_destroy(t);
```
Browsing objects

Always remember that hwloc's basic representation of the machine is a tree, but it also has levels.
Browsing objects

Socket
- depth = 1
- logical_index = 0
- os_index = 0
- sibling_rank = 0
- arity = 2

Cache
- depth = 2
- logical_index = 1
- os_index = 1
- sibling_rank = 1
- arity = 1

Cache
- depth = 2
- logical_index = 2
- os_index = 1
- sibling_rank = 0
- arity = 1

Core
- depth = 3
- logical_index = 1
- os_index = 1
- sibling_rank = 0
- arity = 1

Core
- depth = 3
- logical_index = 2
- os_index = 0
- sibling_rank = 0
- arity = 1

Core
- depth = 3
- logical_index = 3
- os_index = 1
- sibling_rank = 0
- arity = 1
Browsing objects

Thus several ways to traverse objects

- **Tree way**

  ```c
  void traverse(hwloc_obj_t obj) {
    work_on(obj);
    for (i=0; i<obj->arity; i++)
      traverse(obj->children[i]);
  }
  traverse(hwloc_get_root_obj(t));
  ```

- **Array way**

  ```c
  for (depth=0; depth<hwloc_topology_get_depth(t); depth++)
    for (i=0; i<hwloc_get_nbobjs_by_depth(t,depth); i++)
      work_on(hwloc_get_obj_by_depth(t, depth, i));
  ```

Or various combinations of both, see `<hwloc/helper.h>` examples
Browsing objects

A **lot** of browsing helpers and examples in `<hwloc/helper.h>`
- `hwloc_get_common_ancestor_obj`
- `hwloc_obj_is_in_subtree`
- `hwloc_get_largest_objs_inside_cpuset`
- `hwloc_get_obj_covering_cpuset`
- `hwloc_get_cache_covering_cpuset`
- `hwloc_get_shared_cache_covering_obj`
- ...

Browsing objects

Accessing devices

• They are on separate levels
  - HWLOC_TYPE_DEPTH_PCI_DEVICE
  - HWLOC_TYPE_DEPTH_OS_DEVICE

• Helpers are provided to access them directly
  - hwloc_get_pcidev_by_busid(topology, domain, bus, dev, fun);
  - hwloc_cuda_get_device_pcidev(topology, cudevice);
  - hwloc_ibv_get_device_osdev_by_name(topology, name);

Look at their source code, they are examples of browsing the tree.
Object information

- obj->type
- obj->cpuset
- obj->father, children, next_cousin, ...

Depending on the type of object
- obj->cache.size
- obj->cache.linesize
- obj->pcidev.linkspeed
- ...

...
CPU/node set manipulations

Bitmap data structure, with all usual operations
hwloc_bitmap_alloc/free/dup/copy
hwloc_bitmap_set/set_range/clr/clr_range
hwloc_bitmap_isset/iszero/isfull
hwloc_bitmap_first/next/last/weight
hwloc_bitmap_foreach_begin/end
hwloc_bitmap_or/and/andnot/xor/not
hwloc_bitmap_intersects/isincluded/isequal/compare
...

CPU binding API

OS support varies
- Process-wide binding, thread binding, strict, …
- ENOSYS returned when not supported

Should be supported mostly everywhere: single-threaded process binding itself
- `hwloc_set_cpubind(t, cpuset, 0);`

Or the thread itself only
- `hwloc_set_cpubind(t, cpuset, HWLOC_CPUBIND_THREAD);`

Another process
- `hwloc_set_proc_cpubind(t, pid, cpuset, 0);`

...
Memory binding API

OS support varies even more
- Binding existing range, migrating allocated memory, allocating bound memory, strict, ...
- ENOSYS returned when not supported
Should be supported mostly everywhere: allocating bound memory, possibly through process policy change
- hwloc_alloc_membind_policy(t, size, cpuset, DEFAULT, 0);

Changing the binding policy for future mallocs and friends
- hwloc_set_membind(t, cpuset, DEFAULT, 0);

Migrating existing range
- hwloc_set_area_membind(t, addr, len, cpuset, DEFAULT, 0);

Whether already-allocated pages are migrated depends on the OS
Hands-on: part 2
http://runtime.bordeaux.inria.fr/hwloc/hwloc_tutorial.html
Conclusion

- Hwloc provides a generic way to represent the machine, and bind processes/threads/memory
- Both command-line tools and API
- Already used by a lot of HPC project, available in most distributions
- Large documentation

What next?

- Plug-in support
  - Automatic network topology discovery
  - Measurement-based discovery
  - X server OS object

http://www.open-mpi.org/projects/hwloc/
Thanks!

www.open-mpi.org/projects/hwloc/