Understanding and managing hardware affinities on hierarchical platforms With Hardware Locality (hwloc)

Brice Goglin – Runtime Team-Project – Inria Bordeaux Sud-Ouest
Agenda

- Quick example as an Introduction
- Bind your processes
- What's the actual problem?
- Introducing hwloc (Hardware Locality)
- Command-line tools
- C Programming API
- Conclusion
Quick example as an Introduction
Machines are increasingly complex
Machines are increasingly complex

- Multiple processor sockets
- Multicore processors
- Simultaneous multithreading
- Shared caches
- NUMA
- GPUs, NICs, ...
  - Close to some sockets (NUIOA)
Example with MPI

• Let's say I have a 64-core AMD machine
  – Not unusual (about 6000$)

• I am running a MPI pingpong between pairs of cores
  – Open MPI 1.6
  – Intel MPI Benchmarks 3.2
Example with MPI (2/3)

- Between cores 0 and 1
  - 1.39 µs latency – 1900MB/s throughput
- Between cores 0 and 4
  - 1.63 µs – 1400 MB/s – Interesting!
- Between cores 0 and 5
  - 0.68 µs – 3600 MB/s – What ?!
- Between cores 0 and 8
  - 1.24 µs – 2400 MB/s
- Between cores 0 and 32
  - 1.34 µs – 2100 MB/s
What is going on
What is going on (2/3)
What is going on (3/3)
Example with MPI (3/3)

- Between cores that share a L2 cache
  - 0.68 µs – 3600 MB/s
- Between cores that only share a L3 cache
  - 1.24 µs – 2400 MB/s
- Between cores inside the same socket
  - 1.34 µs – 2100 MB/s
- Between cores of another socket
  - 1.39 µs – 1900 MB/s
- Between cores of another socket further away
  - 1.63 µs – 1400 MB/s
Ok, what about Intel machines?

- Less hierarchy levels
  - 4 vs 3
  - HyperThreading?

- But same problems
First take away messages

• Locality matters to communication performance
  – Machines are really far from flat
• Cores/processors numbering is crazy
  – Never expect anything sane here
2 Bind your processes
Where does locality actually matter?

- MPI communication between processes on the same node
- Shared-memory too (threads, OpenMP, etc)
  - Synchronization
    - Barriers use caches and memory too
  - Concurrent access to shared buffers
    - Producer-consumer, etc
- 10 years ago, locality was mostly an issue for large NUMA SMP machines (SGI, etc)
  - Today it's everywhere
    - Because multicores and NUMA are everywhere
What to do about locality?

• Place processes/tasks according to their affinities
  • If two tasks communicate/synchronize/share a lot, keep them close
• Adapt your algorithms to the locality
  • Adapt communication/synchronization implementations to the topology
    • Ex: hierarchical barriers
Process binding

- Some MPI implementations bind processes by default (Intel MPI, Open MPI 1.8)
  - Because it's better for reproducibility
- Some don't
  - Because it may hurt your application
    - Oversubscribing?
- Binding doesn't guarantee that your processes are optimally placed
  - It just means your process won't move
    - No migration, less cache issues, etc
To bind or not to bind?

Zeus MHD Blast

No process binding
Process binding

Zeus MHD Blast. 64 Processes/Cores. Mvapich2 1.8. + ICC
Where to bind?

- Default binding strategies?
  - By core first:
    - One process per core on first node, then one process per on second node, ...
  - By node first:
    - One process on first core of each node, then one process on second core on each node, ...

- Your application likely prefers one to the other
  - Usually the first one
    - Because you often communicate with nearby ranks
Binding strategy impact

![Graph showing the impact of binding strategies on execution time over iterations. The x-axis represents the number of iterations, and the y-axis represents execution time in seconds. Three lines are shown: No binding, Binding by Node first, and Binding by Core first.]
How to bind in MPI?

- MPI standard says nothing
- Manually
  - `mpiexec`  
    - `np 1 -H node1 numactl --physcpubind 0 ./myprogram` :
    - `np 1 -H node1 numactl --physcpubind 1 ./myprogram` :
    - `np 1 -H node2 numactl --physcpubind 0 ./myprogram`
  - Rank files, etc
How to bind in MPI? (2/2)

- **Open MPI**
  - `mpiexec --bind-to core --map-by core ...
    - Map by core
  - `Mpiexec --bind-to-core --mca rmaps_lama_map nsc ...
    - Map by node, then by socket, then by core
  - See `mpiexec --help`

- **MPICH**
  - `mpiexec -bind-to core -map-by BSC ...
    - Map by node (Board), then by socket, then by core
  - See `mpiexec -bind-to help`
How to bind in OpenMP? (more later)

- Intel Compiler
  - KMP_AFFINITY=scatter or compact
- GCC
  - GOMP_CPU_AFFINITY=1,3,5,2,4,6
How do I choose?

- **Dilemma**
  - Use cores 0 & 1 to share cache and improve synchronization cost?
  - Use core 0 & 2 to maximize memory bandwidth?
- **Depends on**
  - The machine structure
  - The application needs
- **Locality-aware is very active research topic**
  - TreeMatch for MPI process placement
    - Based on communication pattern
  - StarPU for task-based scheduling
    - Based on history
  - Many others
What's the actual problem?
Example of dual Nehalem Xeon machine
Another example of dual Nehalem Xeon machine
Processor and core numbers are crazy

- Resources ordering is unpredictable
  - Ordered by any combination of NUMA/socket/core/hyperthread
  - Can change with the vendor, the BIOS version, etc
- Some resources may be unavailable
  - Batch schedulers can give only parts of machines
    - Core numbers may be non-consecutive, non starting at 0, etc
- Don't assume anything about indexes
  - Don't use these indexes
    - Or you won't be portable
Level ordering isn't much better

- Intel is usually
  - Machine
  - Socket = NUMA = L3
  - Core = L1 = L2
  - Hyperthread (PU)
Level ordering isn't much better (2/3)

- AMD is different
  - Machine
  - Socket
  - NUMA = L3
  - L2 = L1i
  - Core = L1d
Level ordering isn't much better (3/3)

- Sometimes there are multiple sockets per NUMA nodes
  - And different levels of caches
- Don't assume anything about level ordering
  - Or (again) you won't be portable
  - e.g.: Intel Compiler OpenMP binding may be wrong on AMD machines
Gathering topology information is difficult

- Lack of generic, uniform interface
  - Operating system specific
    - /proc and /sys on Linux
    - rset, sysctl, lgrp, kstat on others
  - Hardware specific
    - x86 cpuid instruction, device-tree, PCI config space, ...
- Evolving technology
  - AMD Bulldozer dual-core compute units
    - It's not two real cores, neither a dual-threaded core
  - New levels? New ordering?
Binding is difficult too

- Lack of generic, uniform interface, again
  - Process/thread binding
    - sched_setaffinity API changed twice on Linux
    - rset, ldom_bind, radset, affinity_set on others
  - Memory binding
    - mbind, migrate_pages, move_pages on Linux
    - rset, mmap, radset, nmadvise, affinity_set on others
  - Different constraints
    - Bind on single core only, on contiguous set of cores, on random sets?
  - Many different policies
Introducing hwloc (Hardware Locality)
What hwloc is

- Detection of hardware resources
  - Processing units (PU), logical processors, hardware threads
    - Everything that can run a task
  - Memory nodes, shared caches
  - Cores, Sockets, … (things that contain multiple PUs)
  - I/O devices
    - PCI devices and corresponding software handles
- Described as a tree
  - Logical resource identification and organization
    - Based on locality
What hwloc is (2/2)

- API and tools to consult the topology
  - Which cores are near this memory node?
  - Give me a single thread in this socket
  - Which memory node is near this GPU?
  - What shared cache size between these cores?
- Without caring about hardware strangeness
  - Non portable and crazy numbers, names, ...
- A portable binding API
  - No more Linux sched_setaffinity API breakage
  - No more tens of different binding API with different types
What hwloc is NOT

- A placement algorithm
  - hwloc gives hardware information
  - You're the one that knows what your software does/needs
  - You're the one that must match software affinities to hardware localities
    - We give you the hardware information you need
- A profiling tool
  - Other tools (e.g. likwid) give you hardware performance counters
    - hwloc can match them with the actual resource organization
History

- Runtime Inria project in Bordeaux, France
  - Thread scheduling over NUMA machines (2003...)
    - Marcel threads, ForestGOMP OpenMP runtime
    - Portable detection of NUMA nodes, cores and threads
      - Linux wasn't that popular on NUMA platforms 10 years ago
        - Other Unixes have good NUMA support
      - Extended to caches, sockets, ... (2007)
  - Raised questions for new topology users
    - MPI process placement (2008)
History

- Marcel's topology detection extracted as standalone library (2009)
- Noticed by the Open MPI community
  - They knew their PLPA library wasn't that good
- Merged both libraries as hwloc (2009)
- BSD-3
- Still mainly developed by Inria Bordeaux
  - Collaboration with Open MPI community
  - Contributions from MPICH, Redhat, IBM, Oracle, ...
Alternative software with advanced topology knowledge

- PLPA (old Open MPI library)
  - Linux specific, no NUMA support, obsolete, dead
- libtopology (IBM)
  - Dead
- Likwid
  - x86 only, needs update for each new processor generation, no extensive C API
    - It's more kind of a performance optimization tool
- Intel Compiler (icc)
  - x86 specific, no API
hwloc's view of the hardware

- Tree of objects
  - Machines, NUMA memory nodes, sockets, caches, cores, threads
    - Logically ordered
  - Grouping similar objects using distances between them
    - Avoids enormous flat topologies
- Many attributes
  - Memory node size
  - Cache type, size, line size, associativity
  - Physical ordering
  - Miscellaneous info, customizable
Using hwloc for this tutorial

- On PlaFRIM, just use
  
  `$ module load hardware/hwloc`

- (and for GPU-related tests)
  
  `$ module load gpu/cuda`

- You may also install it on your local machine
  
  - It will make remote machine consulting easier
Installing hwloc

- Packages available in Debian, Ubuntu, Redhat, Fedora, CentOS, ArchLinux, NetBSD

- You want the development headers too
  - libhwloc-dev, hwloc-devel, ...
Manual installation

- Take a recent tarball at http://www.open-mpi.org/projects/hwloc
- Dependencies
  - On Linux, numactl/libnuma development headers
  - Cairo headers for lstopo graphics
- ./configure --prefix=$PWD/install
  - Very few configure options
- Check the summary at the end of configure
Manual installation

- make
- make install
- Useful environment variables
  - `export PATH=$PATH:<prefix>/bin`
  - `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<prefix>/lib`
  - `export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:<prefix>/lib/pkgconfig`
  - `export MANPATH=$MANPATH:<prefix>/share/man`
Using hwloc

- Many hwloc command-line tools
  - lstopo and hwloc-*
- ... but the actual hwloc power is in the C API
- Perl and Python bindings
5 Command-line Tools
Istopo
(displaying topologies)

Machine (3828MB)
- Socket L#0 + L3 L#0 (4096KB)
  - L2 L#0 (256KB) + Core L#0
    - PU L#0 (P#0)
    - PU L#1 (P#2)
  - L2 L#1 (256KB) + Core L#1
    - PU L#2 (P#1)
    - PU L#3 (P#3)
- HostBridge L#0
- PCI 8086:0046
  - GPU L#0 "controlD64"
- PCI 8086:10ea
  - Net L#2 "eth0"
- PCIBridge
  - PCI 8086:422b
    - Net L#3 "wlan0"
- PCI 8086:3b2f
  - Block L#4 "sda"
  - Block L#5 "sr0"
Istopo

- Many output formats
  - Text, Cairo (PDF, PNG, SVG, PS), Xfig, ncurses
    - Automatically guessed from the file extension
- XML dump/reload
  - Faster, convenient for remote debugging
- Configuration options for nice figures for papers
  - Horizontal/Vertical placement
  - Legend
  - Ignoring things
  - Creating fake topologies
Istopo

$ istopo
$ istopo --no-io -
$ istopo myfile.png
$ ssh host istopo saved.xml
$ istopo -i saved.xml
$ ssh myhost istopo -.xml | istopo --if xml -i -
$ istopo -i “node:4 socket:2 core:2 pu:2”
hwloc-bind
(binding processes, threads and memory)

• Bind a process to a given set of CPUs
  $ hwloc-bind socket:1 -- mycommand myargs...
  $ hwloc-bind os=mlx4_0 -- mympiprogram ...

• Bind an existing process
  $ hwloc-bind --pid 1234 node:0

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

• Find out if a process is already bound
  $ hwloc-bind --get --pid 1234
  $ hwloc-ps
hwloc-calc
(calculating with objects)

• Convert between ways to designate sets of CPUs, objects... and combine them
  $ hwloc-calc socket:1.core:1 ~pu:even 0x00000008
  $ hwloc-calc --number-of core node:0 2
  $ hwloc-calc --intersect pu socket:1 2,3

• The result may be passed to other tools
• Multiple invocations may be combined
• I/O devices also supported
  $ hwloc-calc os=eth0
Other tools

- Get some object information
  - hwloc-info (v1.7+)
- Generate bitmaps for distributing multiple processes on a topology
  - hwloc-distrib
- Save a Linux node topology info for debugging
  - hwloc-gather-topology
- Manipulating multiple topologies, etc.
Hands-on lstopo

- Gather the topology of one server
- Display it on another machine
- Hide caches
- Remove the legend
- Restrict the display to a single socket
- Export to PDF
Hands-on hwloc-bind and hwloc-calc

• Bind a process to a core and verify its binding
• Find the DMA difference between a GPU and both NUMA nodes
  • Measured with 
    /opt/cluster/gpu/cuda/latest/sdk/C/bin/linux/release/bandwidthTest –memory=pinned --device=N
• Find out how many cores are in the second NUMA node
• Find out which cores are close to InfiniBand
• Find out the physical numbers of all non-first hyperthreads
6 C Programming API
API basics

- A hwloc program looks like this

```c
#include <hwloc.h>

hwloc_topology_t topo;

hwloc_topology_init(&topo);
/* ... configure what topology to build ... */
hwloc_topology_load(topo);

/* ... play with the topology ... */

hwloc_topology_destroy(topo);
```
Major hwloc types

- The topology context: `hwloc_topology_t`
  - You always need one
- The main hwloc object: `hwloc_obj_t`
  - That's where the actual info is
  - The structure isn't opaque
    - It contains many pointers to ease traversal
- Object type: `hwloc_obj_type_t`
  - `HWLOC_OBJ PU`, `_CORE`, `_NODE`, ...
Object information

- Type
- Optional name string
- Indexes (see later)
- cpusets and nodesets (see later)
- Tree pointers (*cousin, *sibling, arity, *child*, parent)
- Type-specific attribute union
  - obj->attr->cache.size
  - obj->attr->pcidev.linkspeed
- String info pairs
Browsing as a tree

• The root is `hwloc_get_root_obj(topo)`

• Objects have children
  • `obj->arity` is the number of children
  • The array of children is `obj->children[]`
  • They are also in a list
    • `obj->first_child`, `obj->last_child`
    • `child->prev_sibling`, `child->next_sibling`
    • NULL-terminated

• The parent is `obj->parent` (or NULL)
Browsing as levels

- The topology is also organized as levels of identical objects
  - Cores, L2d Caches, ...
  - All PUs at the bottom
- Number of levels `hwloc_topology_get_depth(topo)`
- Number of objects on a level
  `hwloc_get_nbobjs_by_type(topo, type)`
  `hwloc_get_nbobjs_by_depth(topo, depth)`
- Convert between depth and type using
  `hwloc_get_type_depth()` or `hwloc_get_depth_type()`
Browsing as levels

• Find objects by level and index
  • `hwloc_get_obj_by_type(topo, type, index)`
  • There are variants taking a depth instead of a type
    • Note: the depth of my child is not always my depth + 1
      • Think of asymmetric topologies

• Iterate over objects of a level
  • Objects at the same levels are also interconnect by prev/next_cousin pointers
    • Don't mix up siblings (children list) and cousins (level)
  • `hwloc_get_next_obj_by_type/depth()`
Hands-on browsing the topology

Starting from basic.c

- Print the number of cores
- Print the type of the common ancestor of cores 0 and 2
- Print the memory size near core 0
- Iterate over all PUs and print their physical numbers
Physical or OS indexes

- obj->os_index
  - The ID given by the OS/hardware
- P#3
  - Default in lstopo graphic mode
  - lstopo -p
- NON PORTABLE
  - Depend on motherboards, BIOS, version, ...
- DON'T USE THEM
Logical indexes

- obj->logical_index
  - The index among an entire level
- L#2
  - Default in lstopo except in graphic mode
  - lstopo -l
- Always represent proximity (depth-first walk)
- PORTABLE
  - Does not depend on OS/BIOS/weather
- That's what you want to use
But I still need OS indexes when binding ?!

- NO !
- Just use hwloc for binding, you won't need physical/OS indexes ever again

- If you want to bind the execution to a core
  - hwloc_set_cpubind(core->cpuset)
    - Other API functions for binding entire processes, single thread, memory, for allocating bound memory, etc.
Bitmap, CPU sets, Node sets

- Generic mask of bits: \texttt{hwloc_bitmap_t}
  - Possibly infinite
  - Opaque, used to describe object contents
    - Which PU are inside this object (\texttt{obj->cpuset})
    - Which NUMA nodes are close to this object (\texttt{obj->nodeset})
  - Can be combined to bind to multiple cores, etc.
    - and, or, xor, not, ...
Hands-on bitmaps and binding

- Bind a process to cores 2 and 4
- Print its binding
- Print where it's actually running
  - Repeat
- Rebind to avoid migrating between cores
  - hwloc_bitmap_singlify()
I/O devices

- Binding tasks near the devices they use improves their data transfer time
  - GPUs, high-performance NICs, InfiniBand, ...
- You cannot bind tasks or memory on these devices
  - But these devices may have interesting attributes
    - Device type, GPU capabilities, embedded memory, link speed, ...
I/O objects

- Some I/O trees are attached to the object they are close to
- PCI device objects
  - Optional I/O bridge objects
- How to match your software handle with a PCI device?
  - OS/Software devices (when known)
    - sda, eth0, ib0, mlx4_0
  - Disabled by default
    - Except in Istopo
Hands-on I/O

$ module load gpu/cuda

Starting from cuda.c

• Find the NUMA node near each CUDA device
Extended attributes

- obj->userdata pointer
  - Your application may store whatever it needs there
  - hwloc won't look at it, it doesn't know what's it contains

- (name,value) info attributes
  - Basic string annotations, hwloc adds some
    - HostName, Kernel Release, CPU Model, PCI Vendor, ...
  - You may add more
Configuring the topology

- Between hwloc_topology_init() and load()
  - hwloc_topology_set_xml(), set_synthetic()
  - hwloc_topology_set_flags(), set_pid()
  - hwloc_topology_ignore_type()

- After hwloc_topology_load()
  - hwloc_topology_restrict()
  - hwloc_topology_insert_misc_object...
Helpers

- hwloc/helper.h contains a lot of helper functions
  - Iterators on levels, children, restricted levels
  - Finding caches
  - Converting between cpusets and nodesets
  - Finding I/O objects
  - And much more
- Use them to avoid rewriting basic functions
- Use them to understand how things work and write what you need
Conclusion
More information

- The documentation
- Related pages
- FAQ
- 3-4 hours tutorials with exercises on the webpage
- README and HACKING in the source
- hwloc-users@open-mpi.org for questions
- hwloc-devel@open-mpi.org for contributing
- hwloc-announce@open-mpi.org for new releases
- [https://git.open-mpi.org/trac/hwloc/](https://git.open-mpi.org/trac/hwloc/) for reporting bugs
Thanks!

Questions?

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

Brice.Goglin@inria.fr