ROME Workshop @ IPDPS
Vancouver

Memory Footprint of Locality Information On Many-Core Platforms

Brice Goglin
Inria Bordeaux – Sud-Ouest – France
2018/05/25
## Locality Matters to HPC Applications

<table>
<thead>
<tr>
<th>Package P#0</th>
<th>Package P#1</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMANode P#0 (32GB)</td>
<td>NUMANode P#1 (32GB)</td>
</tr>
<tr>
<td>L3 (135MB)</td>
<td>L3 (135MB)</td>
</tr>
<tr>
<td>L2 (256KB)</td>
<td>L2 (256KB)</td>
</tr>
<tr>
<td>L1d (32KB)</td>
<td>L1d (32KB)</td>
</tr>
<tr>
<td>Core P#0</td>
<td>Core P#1</td>
</tr>
<tr>
<td>PU P#0</td>
<td>PU P#2</td>
</tr>
<tr>
<td>Core P#2</td>
<td>Core P#3</td>
</tr>
<tr>
<td>PU P#4</td>
<td>PU P#6</td>
</tr>
<tr>
<td>Core P#8</td>
<td>Core P#10</td>
</tr>
<tr>
<td>PU P#8</td>
<td>PU P#10</td>
</tr>
<tr>
<td>Core P#11</td>
<td>Core P#12</td>
</tr>
<tr>
<td>PU P#12</td>
<td>PU P#14</td>
</tr>
</tbody>
</table>

## ROME 2018
Locality Matters for I/O too
Who Needs Locality in the HPC Stack?

Front-end Node

- Resource Manager
- Process Launcher

Gather Topology

Launch MPI Ranks

Daemon

Compute Nodes

- Process
  - Kernels
  - Threads
  - Runtime
  - MPI Lib

Process
# Memory in HPC Platforms

## Top500 – 2017/11

<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>Cores</th>
<th>Memory</th>
<th>GB per Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Sunway TaihuLight</td>
<td>10 649 600</td>
<td>1.31 PB</td>
<td>0.12</td>
</tr>
<tr>
<td>#2</td>
<td>Tianhe-2</td>
<td>3 120 000</td>
<td>1 PB</td>
<td>0.32</td>
</tr>
<tr>
<td>#3</td>
<td>Piz Daint</td>
<td>361 760</td>
<td>340 TB</td>
<td>1.06</td>
</tr>
<tr>
<td>#4</td>
<td>Gyoukou</td>
<td>19 860 000</td>
<td>575 TB</td>
<td>0.028</td>
</tr>
<tr>
<td>#5</td>
<td>Titan</td>
<td>560 640</td>
<td>710 TB</td>
<td>1.27</td>
</tr>
<tr>
<td>#6</td>
<td>Sequoia</td>
<td>1 572 864</td>
<td>1.5 PB</td>
<td>1</td>
</tr>
<tr>
<td>#7</td>
<td>Trinity</td>
<td>301 056 + 678 912</td>
<td>2 PB</td>
<td>2</td>
</tr>
<tr>
<td>#8</td>
<td>Cori</td>
<td>622 336</td>
<td>878 TB</td>
<td>1.41</td>
</tr>
<tr>
<td>#9</td>
<td>Oakforest-PACS</td>
<td>556 104</td>
<td>919 TB</td>
<td>1.65</td>
</tr>
<tr>
<td>#10</td>
<td>K-computer</td>
<td>705 024</td>
<td>1.4 PB</td>
<td>2</td>
</tr>
</tbody>
</table>
hwloc’s Modeling of Platforms

- Tree of hierarchical resource objects (hwloc_obj structure)
  - With many attributes
    - Location with respect to CPU and memory resources (bitmaps)
    - Indexes
    - Links to parent, children, siblings, cousins
    - Type-specific attributes
      - Amount of memory, kind of cache, etc.
    - Strings for custom attributes
      - CPU model, MAC address, name, PCI vendor, etc.

- A little bit of system-wide info
  - hwloc_topology structure
hwloc Memory Footprint on KNL

- Between 400 and 500 objects
  - 256 hwthreads (PUs), 64 caches per level, 64 cores
  - Between 1 and 8 NUMA nodes
  - Some I/O objects
- About 700kB total
- Some users complain
  - They use many processes per node
  - They want to keep that memory available for the application
    - Even if it’s about 0.1 percent of the available memory per core
  - Things will get worse in the future
Horizontal Filtering of Available Resources

Only part of the platform is available to each job
Vertical Filtering of Useful Resources
Some levels aren’t useful

Machine

Package

Package

LLC

L2

L2

L2

L1

L1

L1

L1

L1

L1

Core

Core

Core

Core

Core

Core

Core

Core

Core
Possible Ways to Manage Multiple Clients

- Native Discovery
  - Expensive, should be performed as rarely as possible (PDP’17)

- XML exchange
  - Much faster
  - Still instantiates multiple topologies in memory

- Centralizing in a server
  - Single instance
  - Requires to redirect process queries to the server
    - Slower, API change?

- Shared memory
Shared Memory, obviously but ...

- hwloc was designed in 2009
  - Many objects attributes, many ways to traverse the topology
    - We decided we didn’t want so maaaaaany accessor functions to manipulate these attributes and pointers
- Many users are tied to the existing API
  - Pointers must remain valid, even if mapped in another process
    - Means all processes must map at the same virtual address
  - Or we would have to replace the entire existing API
The Virtual Address Space is mostly empty

- 128TB of VA on current x86 platforms
  - 64PB on next-generation (Intel ia57 extension)
- Similar values on ARM64 and Power
- The available per-core physical memory is **MUCH** lower (GB)

- Trinity/KNL (96GB/node)
  - 99.925% of VM free if one process per node
  - 99.9988% if one process per core
- Summit/P9 (512GB/node) 99.2% and 99.981% respectively
## Virtual Address Space Layout on Linux

<table>
<thead>
<tr>
<th>Stack</th>
<th>Library 1</th>
<th>Library 2</th>
<th>File Mapping</th>
<th>Library 3</th>
<th>Free Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt;100TB</td>
</tr>
</tbody>
</table>

- stack
- library 1
- library 2
- file mapping
- library 3
- Free Space >100TB

<table>
<thead>
<tr>
<th>Stack</th>
<th>Library 1</th>
<th>Library 2</th>
<th>File Mapping</th>
<th>Library 3</th>
<th>Free Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt;100TB</td>
</tr>
</tbody>
</table>

- stack
- library 1
- library 2
- file mapping
- library 3
- Free Space >100TB

<table>
<thead>
<tr>
<th>Stack</th>
<th>Library 1</th>
<th>Library 2</th>
<th>File Mapping</th>
<th>Library 3</th>
<th>Free Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt;100TB</td>
</tr>
</tbody>
</table>

- stack
- library 1
- library 2
- file mapping
- library 3
- Free Space >100TB

### Candidate Location for Shmem Topology

- stack
- library 1
- library 2
- file mapping
- library 3
- Free Space >100TB

- stack
- library 1
- library 2
- file mapping
- library 3
- Free Space >100TB

- stack
- library 1
- library 2
- file mapping
- library 3
- Free Space >100TB

- stack
- library 1
- library 2
- file mapping
- library 3
- Free Space >100TB

---

*ROME 2018*
Implementation in Open MPI

- One ORTE daemon per node
  - Finds the largest hole in its own virtual address space
    - Doesn’t know what other processes will look like
  - Allocates a shared memory region there
  - Stores the hwloc topology in it
- MPI ranks map that shared region
  - Use the hwloc topology contained there
  - If mapping failed (e.g. virtual address range not available)
    - Fall back to XML as usual
hwloc shmem topology

hwloc_shmem_topology_write()

hwloc_shmem_topology_adopt()

Master

Shared Memory

Slave

attrs

obj3

obj2

obj1

shmem clone

attrs

obj3

obj2

topology

obj1

libhwloc.so

adopted
topology

libhwloc.so

libhwloc.so
Experimentation Platforms

- KNL64 = 430 hwloc objects
  - Intel Xeon Phi 7230 (64 cores, 1.3GHz)
  - SNC-4, Flat

- NUMA96 = 405 hwloc objects
  - 4x Intel Xeon E7-8890v4 (24 cores each, 2.2GHz)
  - Cluster-on-Die, no Hyper-threading

- Normal24 = 97 objects
  - 2x Intel Xeon E5-2680v3 (12 cores each, 2.5GHz)
  - Cluster-on-Die, no Hyper-threading
Memory Footprint per MPI rank

- ORTE hello instrumented with mallinfo

<table>
<thead>
<tr>
<th></th>
<th>Native Discovery</th>
<th>XML</th>
<th>Shared-Memory</th>
<th>No topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNL64</td>
<td>2.21MiB</td>
<td>2.35MiB</td>
<td>1.614MiB</td>
<td>1.613MiB</td>
</tr>
<tr>
<td>NUMA96</td>
<td>1.82MiB</td>
<td>1.94MiB</td>
<td>1.230MiB</td>
<td>1.229MiB</td>
</tr>
<tr>
<td>Normal24</td>
<td>1.74MiB</td>
<td>1.78MiB</td>
<td>1.535MiB</td>
<td>1.534MiB</td>
</tr>
</tbody>
</table>
Memory Saving per Node

<table>
<thead>
<tr>
<th>System</th>
<th>Processes</th>
<th>Saving per Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNL64</td>
<td>1 process</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2 processes</td>
<td>500kiB</td>
</tr>
<tr>
<td></td>
<td>64 processes (1 per core)</td>
<td>31MiB</td>
</tr>
<tr>
<td></td>
<td>256 processes (1 per hwthread)</td>
<td>127MiB</td>
</tr>
<tr>
<td>NUMA96</td>
<td>1 process</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2 processes</td>
<td>590kiB</td>
</tr>
<tr>
<td></td>
<td>96 processes (1 per core)</td>
<td>56MiB</td>
</tr>
<tr>
<td>Normal24</td>
<td>1 process</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2 processes</td>
<td>200kiB</td>
</tr>
<tr>
<td></td>
<td>24 processes (1 per core)</td>
<td>4.6MiB</td>
</tr>
</tbody>
</table>
## Launch Time

<table>
<thead>
<tr>
<th></th>
<th>Native Discovery</th>
<th>XML</th>
<th>Shared-Memory</th>
<th>Speedup vs XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNL64 – 64 procs</td>
<td>9.69s</td>
<td>4.16s</td>
<td>1.68s</td>
<td>x 2.48</td>
</tr>
<tr>
<td>KNL64 – 256 procs</td>
<td>47.20s</td>
<td>18.45s</td>
<td>7.02s</td>
<td>x 2.63</td>
</tr>
<tr>
<td>NUMA96 – 96 procs</td>
<td>7.29s</td>
<td>1.17s</td>
<td>0.56s</td>
<td>x 2.10</td>
</tr>
<tr>
<td>Normal24 – 24 procs</td>
<td>0.84s</td>
<td>0.53s</td>
<td>0.47s</td>
<td>x 1.13</td>
</tr>
</tbody>
</table>
Launch Time – KNL64

Launch Time (Seconds).

Number of Processes.

Native Discovery
XML
Shared-Memory
Conclusion

● Many components of the HPC stack use topology information
  • And many processes per node
● Must share topology information to reduce memory footprint
  • Already needed on many-core platforms
Contribution

- hwloc may now place topology in shared-memory
- We designed a way to use the same virtual address in all processes
  - Required to maintain compatibility with old hwloc API
- Available in Open MPI 4.0 and hwloc 2.0
- Reduces footprint to a single topology per node
- Reduces launch time significantly
Future Work

- Share topology between jobs with different sets of allocated resources on same node
- Extend to other process managers
  - Slurm’s srun, etc.
- Propagate shared topology information to all layers inside each process
  - Cooperation between MPI, OpenMP, etc.
Thank you for your attention

Brice.Goglin@inria.fr