NUMA Kernel APIs

Dimitri Korsch

NUMA Seminar | 03.12.2014

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Agenda

1. Solaris
2. Linux
3. Windows
4. Portable Hardware Locality - hwloc

0. What is a "Node"?

node = at least one processor and associated local memory

Solaris [1]

Windows [3]

node = "Each node has its own processors and memory …"

node = all memory has the same speed as seen from a particular set of CPUs

Linux [2]

1. Solaris
locality groups

- Locality groups (**`lgrp`**): processor and memory resources
- **`lgrp`** is an hierarchical directed acyclic graph (DAG)
- **`lgrps`** are enumerated with respect to the root node of the DAG
- LIVE DEMO: **`lgrpinfo`**

DEFINITION:
node = at least one processor and associated local memory [1]

```
lgroup 0 (root):
        Children: 1 2
        CPUs: 0-7
        Memory: installed 16G, allocated 3.8G, free 12G
        Lgroup resources: 1 2 (CPU); 1 2 (memory)
        Latency: 90
lgroup 1 (leaf):
        Children: none, Parent: 0
        CPUs: 0-3
        Memory: installed 8.0G, allocated 1.8G, free 6.2G
        Lgroup resources: 1 (CPU); 1 (memory)
        Load: 0.263
        Latency: 54
lgroup 2 (leaf):
        Children: none, Parent: 0
        CPUs: 4-7
        Memory: installed 8.0G, allocated 2.0G, free 6.0G
        Lgroup resources: 2 (CPU); 2 (memory)
        Load:    0
        Latency: 54
```

1. Solaris
memory placement

- 2 Modes:
  - next-touch
    - next thread which touches a specific block of memory will possibly have access to it locally i.e. if remote memory is accessed it will possibly be migrated
    - default for thread private data
  - random
    - Memory is placed randomly amongst the **`lgrps`**
    - useful for shared memory regions accessed by multiple threads
- **placement verification:**
  - variety of tools to monitor process and thread **`lgrp`** mappings

1. Solaris

kernel API: liblgrp (#include <sys/lgrp_user.h>)

- lgrp information
- provide memory management hints to the OS
- **`madvise`**`()`:

  memory placement advice to kernels virtual memory manager
- **`meminfo`**`()`:

  virtual to physical memory mapping information
- 3 levels of thread affinity: strong, weak or none
- memory placement is determined
  - firstly by the allocation policy
  - then with respect to threads accessing it
- NO direct API for allocating memory to specific **`lgrp`**
- bind thread ot specific processor with **`processor_bind`**`()`

- support since 2.5 Linux Kernel
- more information from Lukas and Fredrik

DEFINITION:
node = all memory has the same speed as seen from a particular set of CPUs [2]

2. Linux
memory management policies

1. local (default)
   - map pages on to the physical node which faulted them in
   - maximizes data locality in many cases
2. strict allocation to a node
   - Memory allocation at a given node, fails if there is not enough
   - memory on the node
3. prefered
   - try on prefered node
   - fallback to default policy
4. interleaved
   - memory is dispersed equally amongst the nodes

2. Linux
kernel API

sys calls to implement different NUMA policies:

- modify **scheduling** and **virtual memory** related variables within the kernel
- **`mbind()`**: set NUMA policy for a specific memory area
- **`set_mempolicy()`**: set policy for a specific process
- **`sched_setaffinity()`**: set process' CPU affinity

⇒ **difficult to use**

2. Linux
kernel API: libnuma (#include <numa.h>) & numactl

numactl

- **command line tool** to control the **NUMA policy** and **CPU placement** of an **entire executable**
- also can be used to display NUMA related hardware configuration and configuration status

libnuma

- user space shared library (**cc <...> -lnuma**)
- usefull functions like **`numa_run_on_node()`** for application programming
- for more information: **`man libnuma`**
- does **NOT** provide a means for determining where a given area of memory is physically located

2. Linux

kernel API: libnuma example

https://gist.github.com/dc8c4283ac2bdae4322e.git

```
if(numa_available() < 0)
    printf("System does not support NUMA API!\n");
int n = numa_max_node();
int size = 1024 * 1024;
printf("There are %d nodes on your system\n", n + 1);
void *mem = numa_alloc_onnode(size, n);
if(mem == NULL)
    printf("could not allocate memory on node %d!\n", n);
numa_free(mem, size);
if (numa_run_on_node(n) != 0)
    printf("could not assign current thread to node %d!\n", n);
```

3. Windows

- Basic terminology:
  - logical processor < core < physical processor
  - processor group = up to 64 logical processors

  ⇒ not supported by XP, Vista, Server 2003, Server 2008

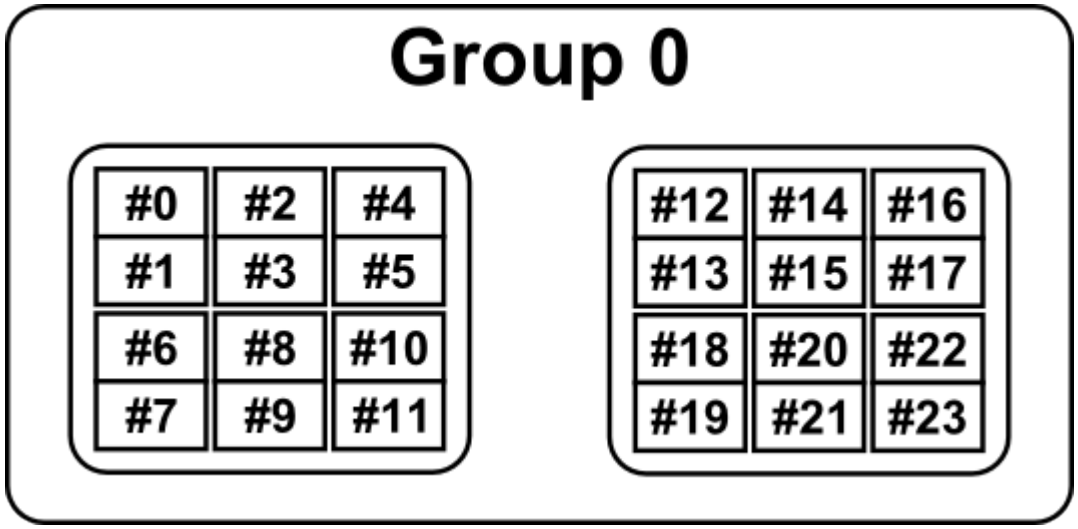
DEFINITION (from Windows online Docs)*:*
node = "Each node has its own processors and memory …" [3]

3. Windows
processor groups

- logical processors are assigned on the start to a group
- system takes physical locality into account when assigning
- nodes are assigned to a single group; multiple nodes in one group possible
- node with more than 64 logical processors ⇒ node is splitted

3. Windows

processor groups

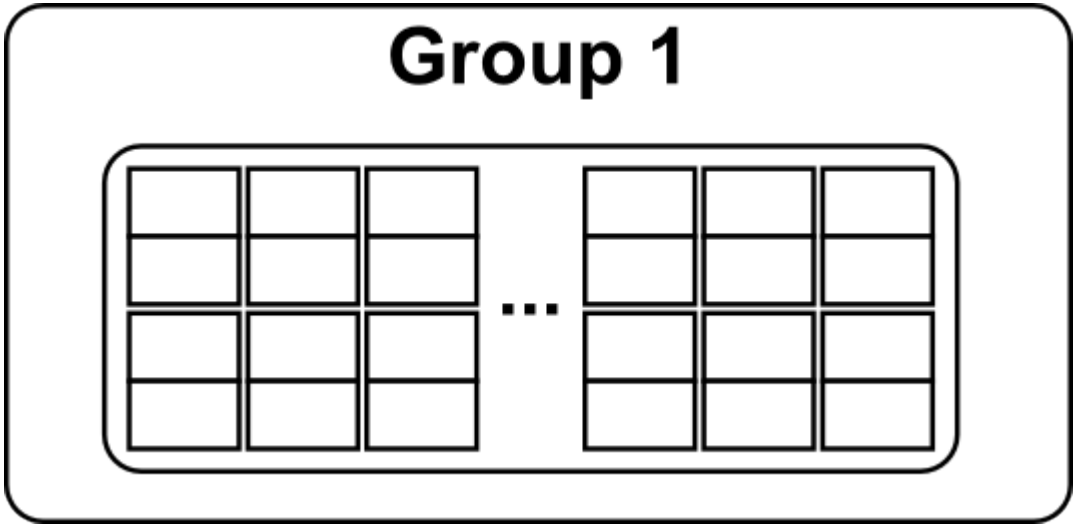


2x Xeon E5-2620

6 Cores / 12 Logical Procs

Xeon Phi

50+ Cores



3. Windows

kernel API (#include <windows.h>) [3]

1. node layout
   - **GetNumaHighestNodeNumber**
   - **GetProcessAffinityMask** - returns list (mask) of processors
   - **GetNumaProcessorNode / -Ex** - returns node of a processor
   - **GetNumaNodeProcessorMask / -Ex** - returns list (mask) of processors in a node
2. set process' / thread's affinities
   - **SetProcessAffinityMask**
   - **SetThreadAffinityMask**
3. allocate memory
   - **GetNumaAvailableMemoryNode / -Ex** - available memory to a node
   - **VirtualAllocExNuma** - specify prefered node

   ⇒ memory allocation on demand; if node out of memory, allocate from other nodes

3. Windows

kernel API example

https://gist.github.com/df6bc5953babc6fd3a7a.git

```
PSYSTEM_LOGICAL_PROCESSOR_INFORMATION buffer = NULL;
GetLogicalProcessorInformation(buffer, &size); // MAGIC
while(offset + sizeof(SYSTEM_LOGICAL_PROCESSOR_INFORMATION) <= size){
  switch (buffer->Relationship){
    case RelationNumaNode: numaNodeCount++; break;
    case RelationProcessorCore: processorCoreCount++;
      logicalProcessorCount += CountSetBits(buffer->ProcessorMask);
      break;
    case RelationCache:
      if (&buffer->Cache->Level == 1) processorL1CacheCount++;
      else if (&buffer->Cache->Level == 2) processorL2CacheCount++;
      else if (&buffer->Cache->Level == 3) processorL3CacheCount++;
      break;
    }
    offset += sizeof(SYSTEM_LOGICAL_PROCESSOR_INFORMATION); buffer++;
}
```

3. Windows

kernel API example

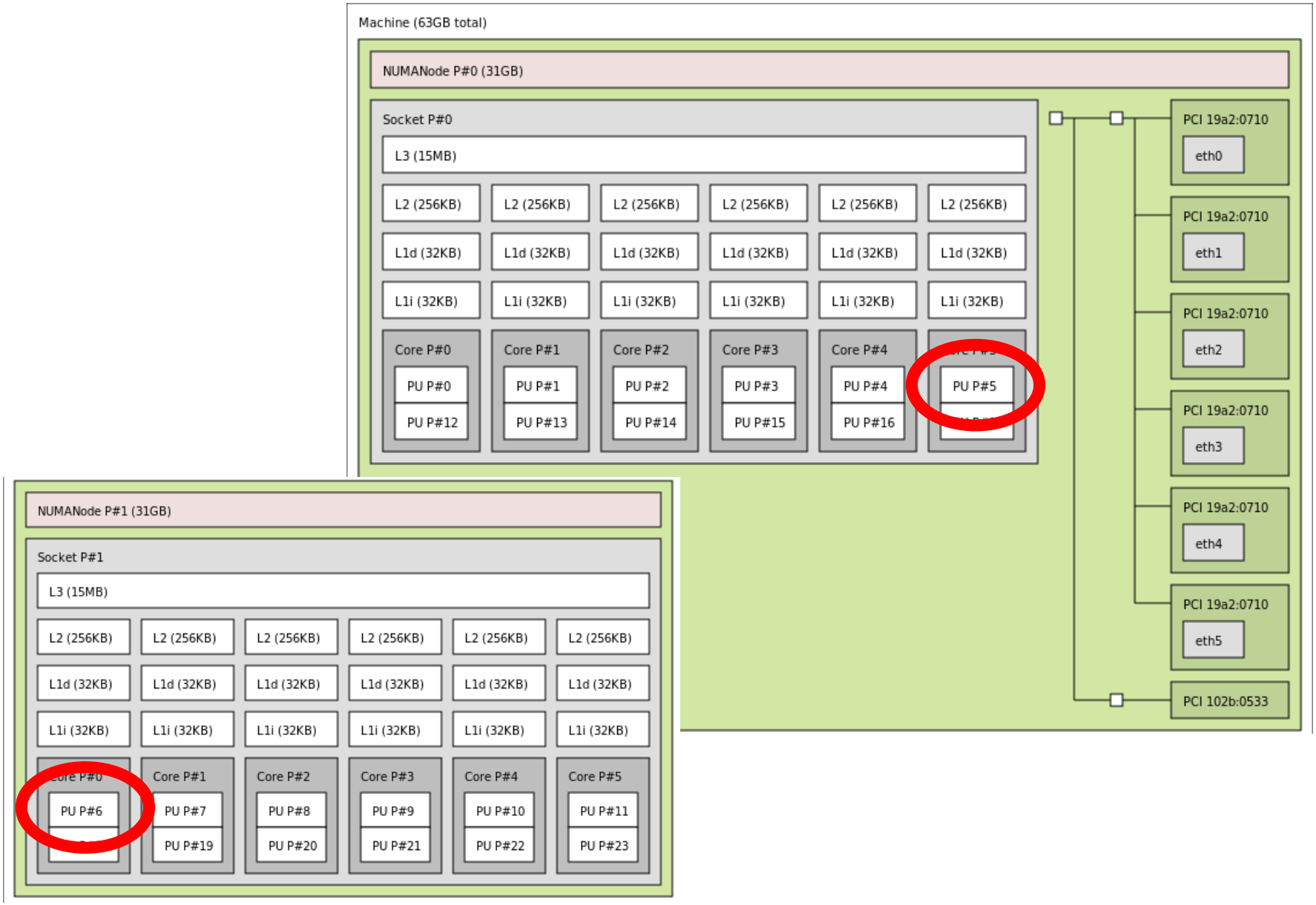
https://gist.github.com/df6bc5953babc6fd3a7a.git

```
GetLogicalProcessorInformation results:
Number of NUMA nodes: 1
Number of processor cores: 2
Number of logical processors: 4
Number of processor L1/L2/L3 caches: 4/2/1
```

4. Portable Hardware Locality - hwloc

- Subproject of OpenMPI group
- **command line tools** and a **C API**
- gathers various attributes such as cache and memory information
- primarily aims at helping high-performance computing applications
- supported operating systems:
  - **Linux**
  - **Microsoft Windows**
  - **Solaris**
  - AIX
  - Darwin / OS X
  - FreeBSD and its variants (such as kFreeBSD/GNU)
  - NetBSD
  - OSF/1 (a.k.a., Tru64)
  - HP-UX
  - IBM BlueGene/Q Compute Node Kernel (CNK)

4. Portable Hardware Locality - hwloc: lstopo



4. Portable Hardware Locality - hwloc

installation (Linux) and compilation

- get sources (http://www.open-mpi.org/projects/hwloc/)
- `./configure --prefix=$HWLOC_HOME`

  `(HWLOC_HOME = ~/hwloc)`
- `make && make install`
- create hwloc_ex.c
- `gcc hwloc_ex.c \`

  `-I$(HWLOC_HOME)/include \`

  `-o hwloc-ex \`

  `-L$(HWLOC_HOME)/lib -lhwloc`

4. Portable Hardware Locality - hwloc

1st example

https://gist.github.com/4c5f9e4b90c5a2276e9b.git

```
/* Allocate and initialize topology object. */
hwloc_topology_t topology;
hwloc_topology_init(&topology);
hwloc_topology_load(topology);
int topodepth = hwloc_topology_get_depth(topology);

>>> [go over each level and show objects(devices)] <<<

int depth = hwloc_get_type_depth(topology, HWLOC_OBJ_SOCKET);
printf("%u sockets\n", hwloc_get_nbobjs_by_depth(topology, depth));
```

4. Portable Hardware Locality - hwloc

2nd example

https://gist.github.com/4c5f9e4b90c5a2276e9b.git

```
/*********************************************************
 * 1. allocate some memory on the last NUMA node
 * 2. bind some existing memory to the last NUMA node.
 *********************************************************/
 int n = hwloc_get_nbobjs_by_type(topology, HWLOC_OBJ_NODE);
 void *m; int size = 1024*1024;
 hwloc_obj_t 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);
```

NUMA Kernel APIs

Dimitri Korsch

NUMA Seminar | 03.12.2014

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Sources

[1] - Antony, Joseph, Pete P. Janes, and Alistair P. Rendell. "Exploring thread and memory placement on NUMA architectures: Solaris and Linux, UltraSPARC/FirePlane and Opteron/HyperTransport." High Performance Computing-HiPC 2006. Springer Berlin Heidelberg, 2006. 338-352. http://cs.anu.edu.au/people/Alistair.Rendell/papers/ThreadAndMemoryPlacement.Springer.pdf

[2] - http://linux.die.net/man/3/numa

[3] - http://msdn.microsoft.com/en-us/library/windows/desktop/aa363804(v=vs.85).aspx

[4] - http://www.open-mpi.org/projects/hwloc/doc/