The affinity model

Gabriele Fatigati

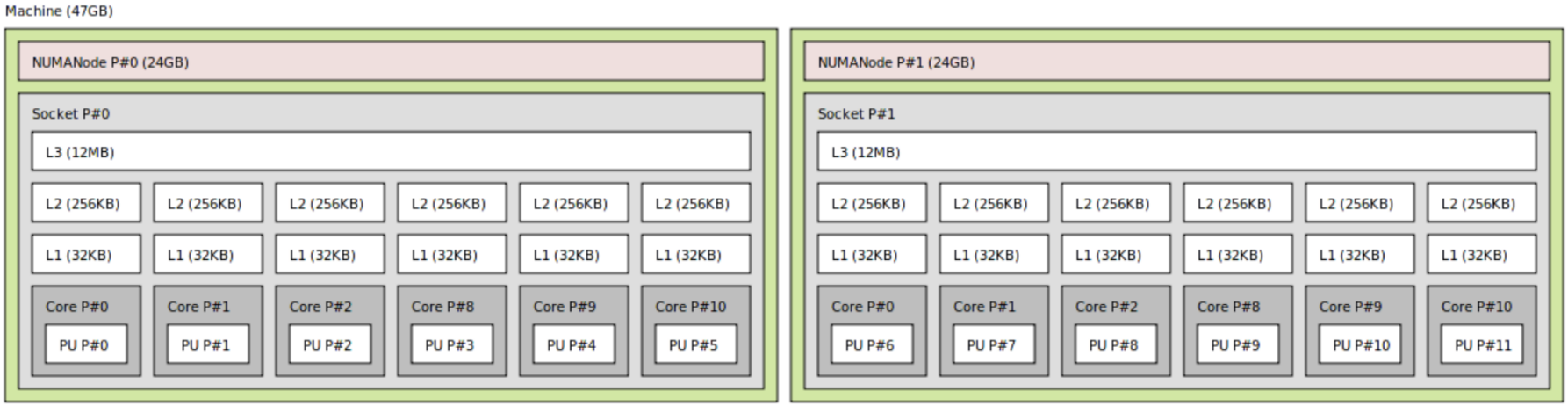
CINECA Supercomputing group
g.fatigati@cineca.it

Outline

NUMA architecture

Definition

NUMA Non-Uniform Memory Access, the memory is divided in local (faster) and remote (slower) areas



- Core 0,1,2,3,4,5 access faster in socket 0 memory area, slower in socket 1 memory area.
- Core 4,5,6,7,8,9 access faster in socket 1 memory area, slower in socket 0 memory area.

Definition

Physical proximity: two cores are physically near when there is a physical interconnection between them. (by BUS, cache or other).
Two threads are near when they are placed on two near cores.

Affinity, what is?

Definition

The affinity is a modification of the native OS queue scheduling algorithm. (rif: wikipedia)

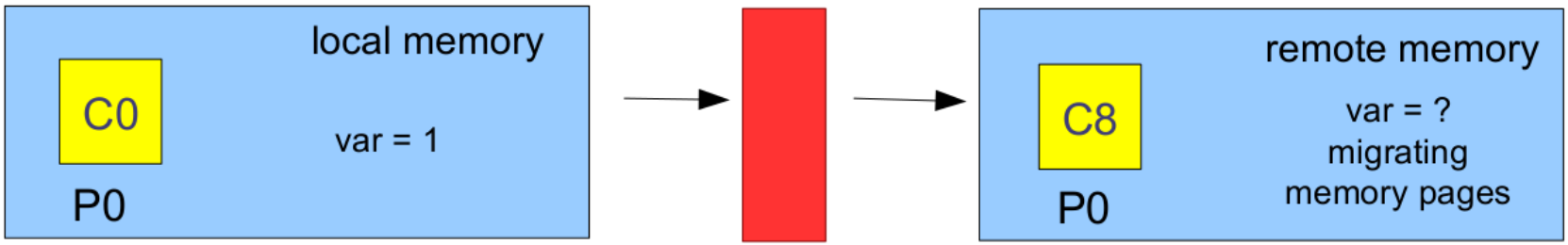
- Execution bind: indicates a preferred core where process/thread will run
- Memory bind: indicates a preferred memory area where memory pages will bound (local areas in NUMA machine)

Migrating process/thread from one core to another, cache and memory locality is lost.

Usually, HPC cluster doesn't have an HPC-oriented kernel, but they have a general purpouse kernel like Linux.

The kernel moves process among cores to optimize the usage of them, assuming that on the node there are many processes besides HPC processes. In non HPC context should be fine, but on HPC cluster can limit the peformance.

In NUMA architecture can be happen also with remote memory:



numactl

Control NUMA policy for processes or shared memory. Runs processes with a specific NUMA scheduling or memory placement policy.

–hardware: Show NUMA policy settings of the current process.

```
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 32733 MB
node 0 free: 17837 MB
node 1 cpus: 8 9 10 11 12 13 14 15
node 1 size: 32767 MB
node 1 free: 1542 MB
node distances:
```

node	0	1
0:	10	20
1:	20	10

- –show policy: default

```
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
cpubind: 0 1
nodebind: 0 1
membind: 0 1
```

This machine has 16 cores, arranged into 2 groups (nodes), with each group of 8 having access to local memory region of 32 GB (64 GB total)

- –cpunodebind=nodes, Only execute process on the CPUs of nodes.

- numactl –cpunodebind=0 –membind=0,1
  Run process on node 0 with memory allocated on node 0 and 1.
- numactl –physcpubind=2
  Run process only on physical CPU having id 2 and allocate the memory on the same NUMA node where the process is running.

- –touch Touch pages to enforce policy early. Default is do not touch them, the policy is applied when an applications maps and accesses to a single page.

  When a processor/thread allocates memory, the memory is not really allocated immediately. The memory pages allocation is done on the first access (usually on the initialization)

```
int*array=(int*)malloc(100*sizeof(int));
// not really allocated

for( int i=0; i \(< \) 100; i++)
      1 array[i] = 0;  // now is allocated
```

Thread affinity and OpenMPI

Is it possible to bind using MPI environment, like OpenMPI:

- -bind-to-core Bind processes to cores.
- -bind-to-socket Bind processes to processor sockets.

The default is do not bind processes.

Binding policies are referred to a node.

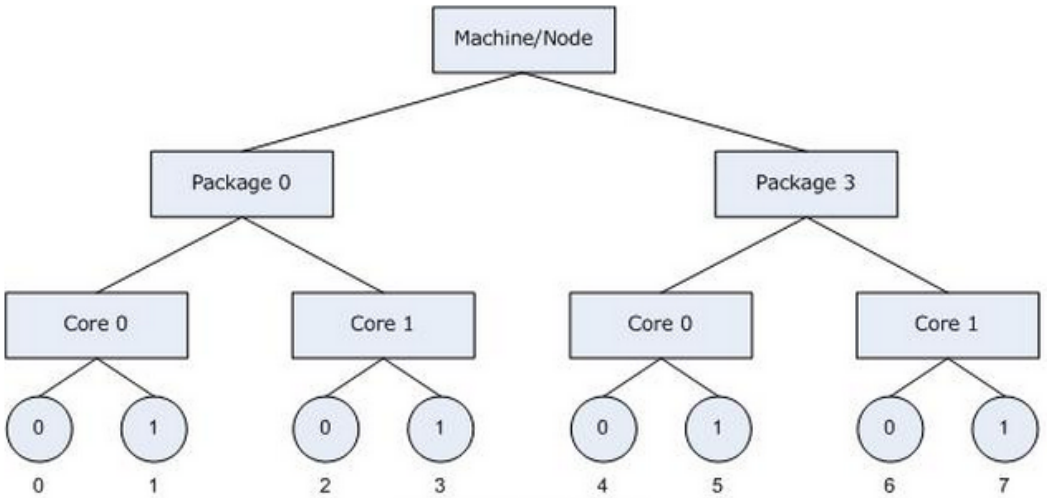
Thread affinity and Intel MPI

Intel MPI can bind threads to physical processing units by using KMP_AFFINITY environment variable.

KMP_AFFINITY=mode

Compact mode

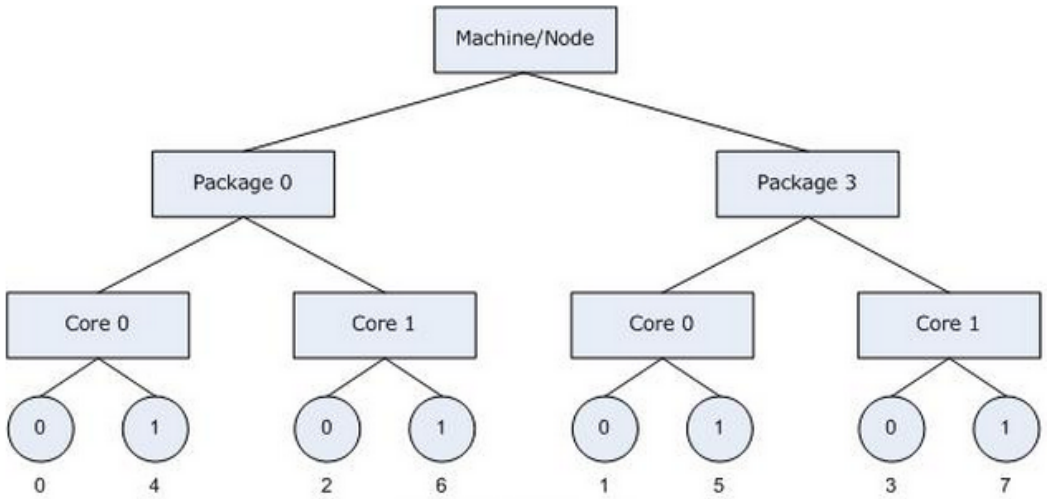
KMP_AFFINITY=compact. Specifying compact assigns the OpenMP thread <n>+1 to a free thread context as close as possible to the thread context where the <n> OpenMP thread was placed. If two threads shares the same data in a cache, put them near can give advantages.



Scatter mode

KMP_AFFINITY=scatter. Specifying scatter distributes the threads as evenly as possible across the entire system. scatter is the opposite of compact

Useful when the cache concurrency bewteen threads is high

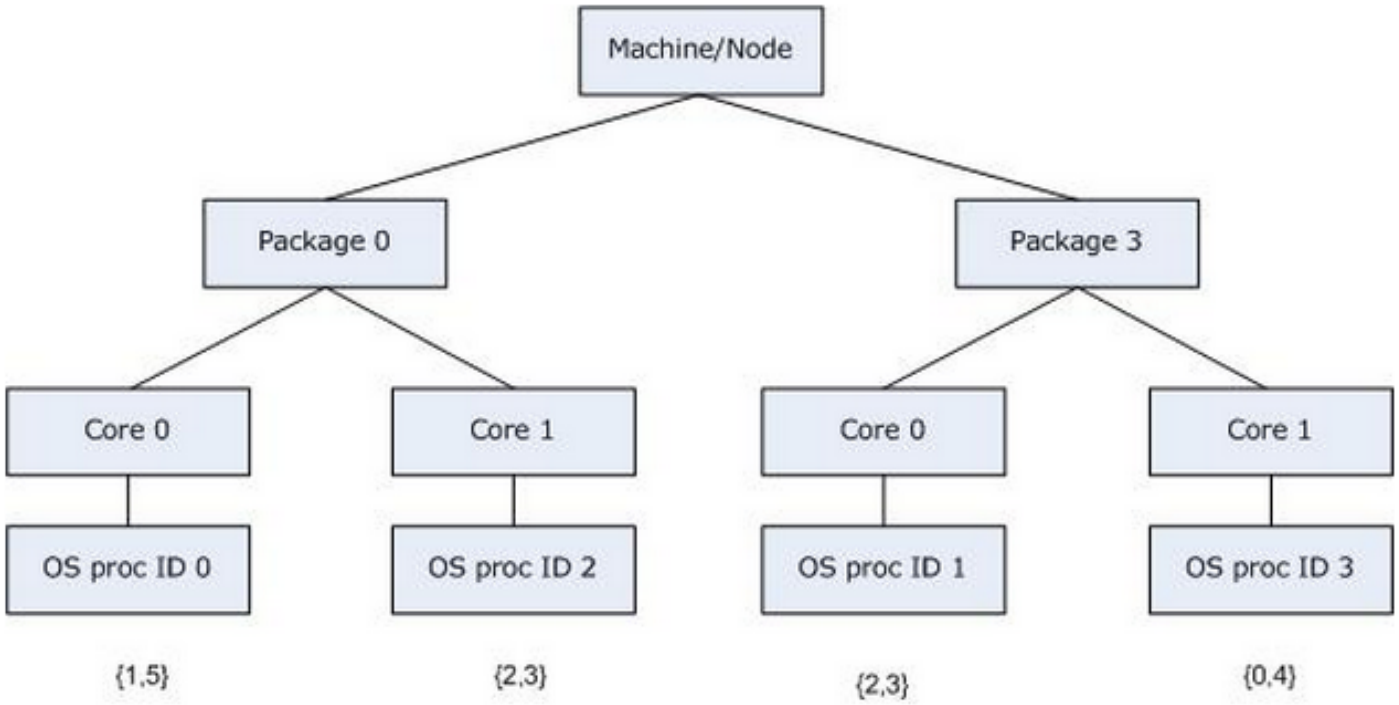


Explicit mode

Instead of allowing the library to detect the hardware topology and automatically assign OpenMP threads to processing elements, the user may explicitly specify the assignment by using a list of operating system (OS) processor (proc) IDs.

However, this requires knowledge of which processing elements the OS proc IDs represent.

KMP_AFFINITY="granularity=fine,proclist=[3,0,{1,2},{1,2}],explicit"



Thread affinity and GCC compiler

On GCC, thread binding is managed by using GOMP_CPU_AFFINITY environment variable (requires OpenMP 3.0 or more)

- GOMP_CPU_AFFINITY="0 3 1-2 4-15:2"
  Bind the initial thread to CPU 0, the second to CPU 3, the third to CPU 1, the fourth to CPU 2, the fifth to CPU 4, the sixth can be placed on CPU 6, 8, 10, 12 or 14

Thread affinity and Linux

sched_setaffinity is a function that sets a processor's affinity to run on a given processor, by using a processors mask.

```
#include <sched.h>
#include <iostream>
#include <stdlib.h>
using namespace std;
int main (int argc, char* argv[])
{
  cpu_set_t mask;
  unsigned int len = sizeof(mask);
  CPU_ZERO(&mask);
  CPU_SET(2,&mask); // set core 2, processor mask = 0010
  sched_setaffinity(getpid(), len, &mask);
  // process will run on CPU 2
}
```

OpenMP standard

Starting from 3.1 version OpenMP implements, a binding procedure throught OMP_PROC_BIND environment variable. If true, the execution environment should not move OpenMP threads between processors. The behaviour is implementation defined. If false, the execution environment may move OpenMP threads between processors.

Thread affinity suggested option: HWLOC

HWLOC stands for HardWare LOCality, library to get many information about the machine like NUMA nodes, shared caches, processor sockets, processor cores and processor units. These information are very useful to bind processor/threads.

Portable on many OS like Linux, AIX, Darwin, Solaris, Windows, FreeBSD.

PU (Processing Unit)

In HWLOC, the PU is The smallest processing element that can be represented by a hwloc object. It may be a single- core processor, a core of a multicore processor, or a single thread in SMT processor.

```
hwloc_topology_t topology;
hwloc_obj_t core;
hwloc_cpuset_t set;
int pu_id = 4;  // process will bound on core 4
hwloc_topology_init(&topology);
hwloc_topology_load(topology);
core = hwloc_get_obj_by_type(machine_topology->topology,
                             HWLOC_OBJ_PU, pu_id);
set = hwloc_bitmap_dup(core->cpuset);
hwloc_bitmap_singlify(set); // keep a single index among those
hwloc_set_cpubind(machine_topology->topology, set, 0);
// some work
hwloc_bitmap_free(set);
```

hwloc_set_cpubind

The most important funcion, used to bind a process/thread to a specific PU

- HWLOC_CPUBIND_PROCESS Bind all threads of the current (possibly) multithreaded process.
- HWLOC_CPUBIND_THREAD Bind current thread of current process.
- HWLOC_CPUBIND_STRICT The process will never execute on other CPUs than the designates CPUs
- HWLOC_CPUBIND_NOMEMBIND Avoid any effect on memory binding.

Memory binding with hwloc

```
hwloc_set_membind(hwloc_topology_t topology,
                  hwloc_const_cpuset_t cpuset,
                  hwloc_membind_policy_t policy,
                  int flags )
```

Bind the already-allocated memory of the current process or thread to prefer the NUMA node near the specified physical cpuset

Memory binding trap

```
/* imagine two thread, the first one bound on node 0,
 the second one allocated on node 1 */

int* array = (int*)malloc(1000*sizeof(int));

#pragma omp parallel
{
 int tid = omp_get_thread_num();
 if(tid==1) {
     for( int i=0; i \(< \) 100; i++)
       array[i] = 0;  // now is intialized
    }
}
```

Array is completely bound on node 1, since thread 1 is bound on it. If thread 0 want to use it, memory pages will be remote so the access is more slow.