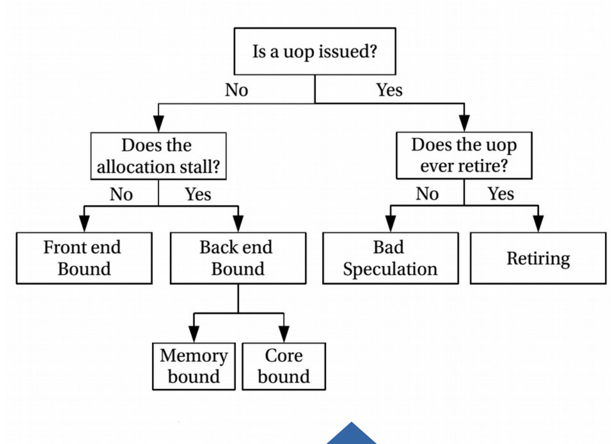
Advanced Optimization Techniques

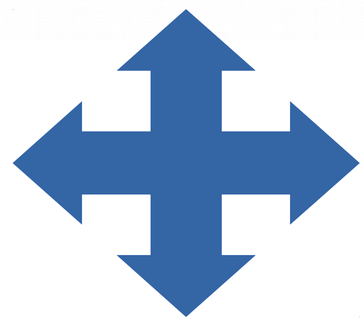
ICTP Trieste 2015

Dr. Christopher Dahnken
Intel GmbH

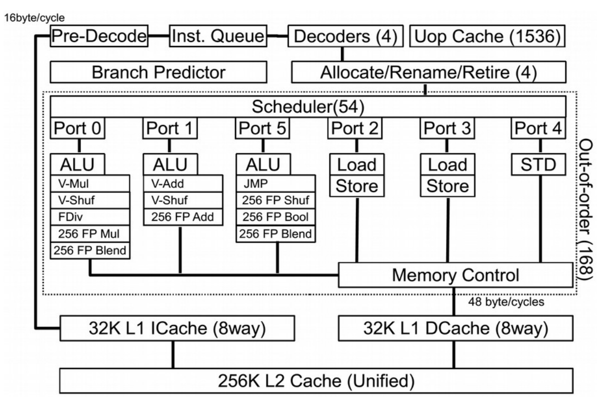
Outline

Method





CPU



Code

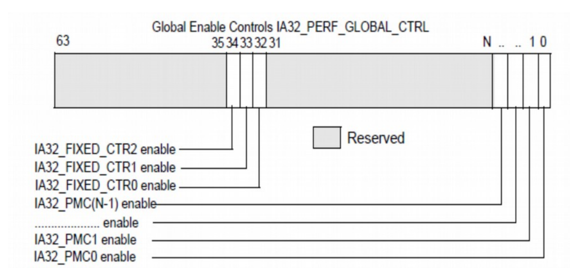
```
!$OMP SECTION
!   tsend=dclock()
 if(iblock.lt.(nblocks)) then
   nxti=m_of_i(iblock+1)
   nxtj=n_of_i(iblock+1)
   nxtk=k_of_i(iblock+1)

   nxt_buffsize_m=buffersize(ms,bm,nxti)
   nxt_index_m=(nxti-1)*bm+1

   nxt_buffsize_n=buffersize(ns,bn,nxtj)
   nxt_index_n=(nxtj-1)*bn+1

   nxt_buffsize_k=buffersize(ks,bk,nxtk)
   nxt_index_k=(nxtk-1)*bk+1
```

Measurement

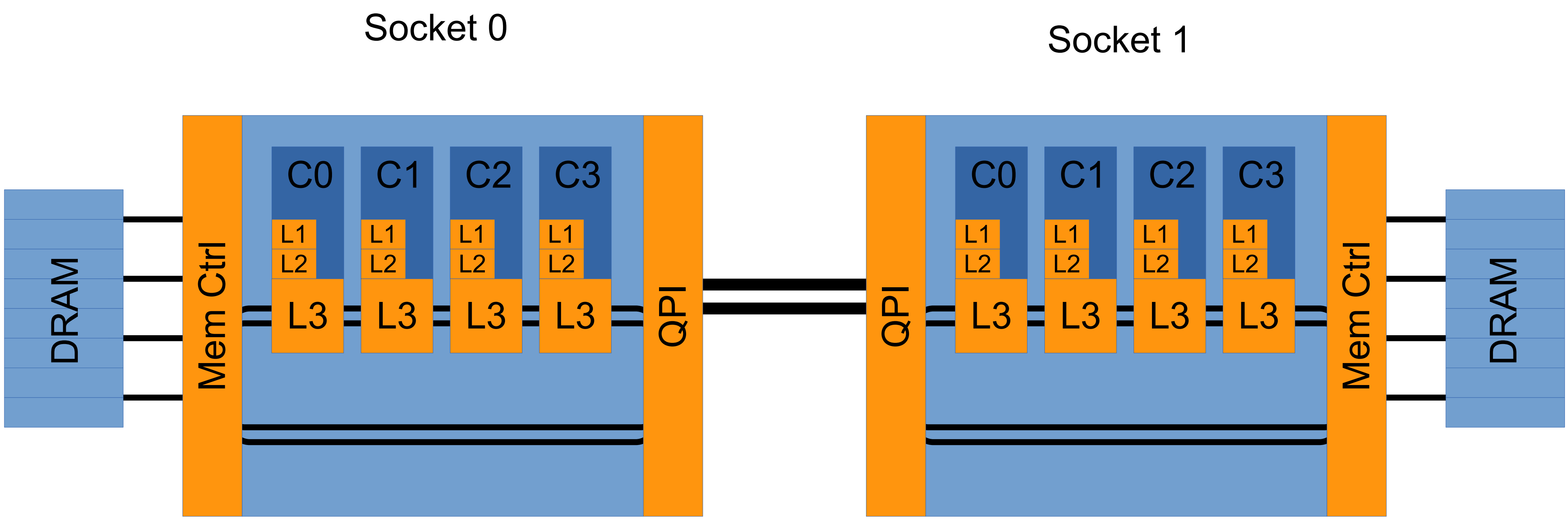


Shared Memory Systems

Intro

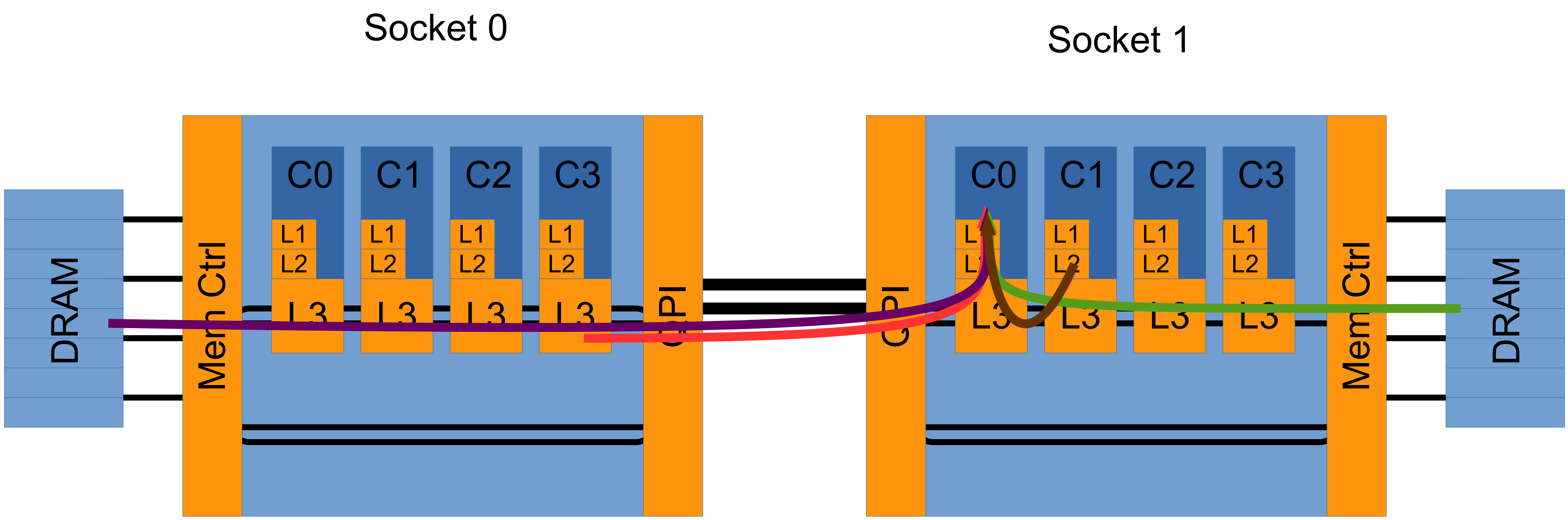
- So far we have only cared about a single core
- Now let's widen this and think of the execution of programs on whole processors and many processors at the same time, but stay in the same address space, i.e. on the same compute server.

Intro – Non Uniform Memory Architecture (NUMA)



Dual socket systems are the main workhorse of HPC today. The complex hierarchy can give rise to various problems: NUMA/UMA, false-sharing, thread placement, latency, bandwidth, etc,etc,etc

Intro

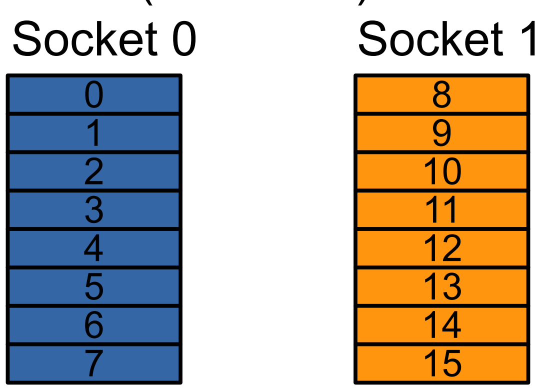


Depending on the location from where a specific piece of memory needs to be transferred from, the time for data provision can vary largely!

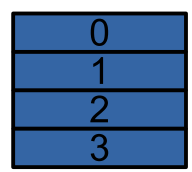
Share memory - NUMA

- Dual socket nodes can operate in two modes

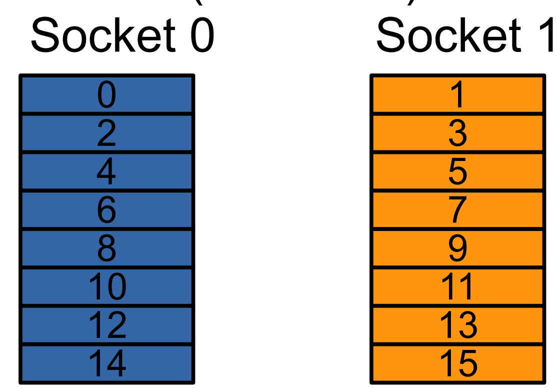
NUMA
(numa=on)



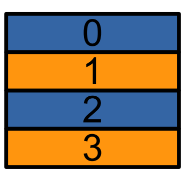
Memory controller maps sockets contiguous memory. Allocated memory might look like this:



Interlaced
(numa=off)



Memory controller maps each other cache-line from another socket to contiguous memory. Allocated memory might look like this:



NUMA mode generally gives the best results for HPC – if you are careful, you will always get the fast local memory performance

Thread and memory placement

Memory placement

- Linux memory is placed where it is first accessed, not where it is allocated (first touch)!

```
   double* a = new double[SIZE];
   for(int i=0;i<SIZE;i++) a[i]=i;
#pragma omp parallel for reduction(+:sum)
   for(int i=0;i<SIZE;i++) {
       sum+=a[i];
   };
```

- This will put all memory on one socket, but potentially read from many (assumed the threads are “pinned”, which will be discussed next)

Thread Placement

- Due to the complicated memory hierarchy (caches, NUMA nodes), it is most important to keep threads on a particular core, so data is readily available when needed.
- Many possible ways:
  - Numactl
  - Taskset
  - OpenMP (KMP_AFFINITY, OMP_PLACES, …)
  - Sched.h (won't talk about this)

Thread placement - taskset

- Taskset is a command of the operating system to manipulate the affinity mask of a process/thread
- It can be used to run commands (set the affinity mask at start time) or change the affinity mask of already running processes

Thread placement - taskset

- Usage:

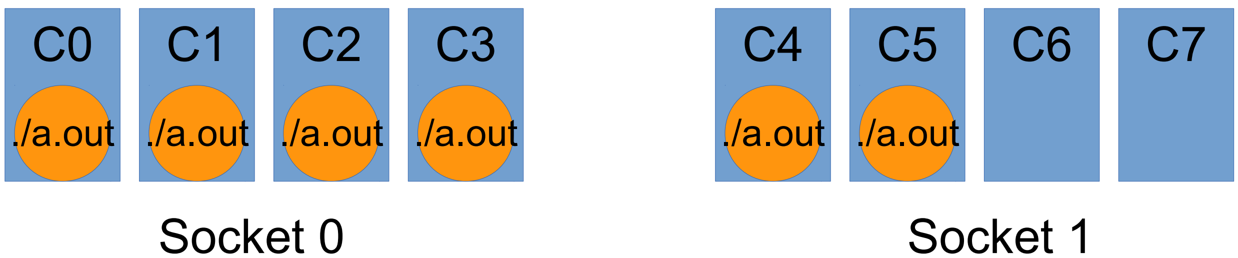
  ```
  $ taskset [-c <list>] <mask> <command>
  $ taskset [-c <list>] -p <mask> <pid>
  ```

- -p <pid>: change the affinity mask of an existing process with PID <pid>
- -c <list>: provide a numerical list of processors instead of an integer mask to set the affinity to, e.g. 0-2,5 would run on cores 0,1,2 and 5.

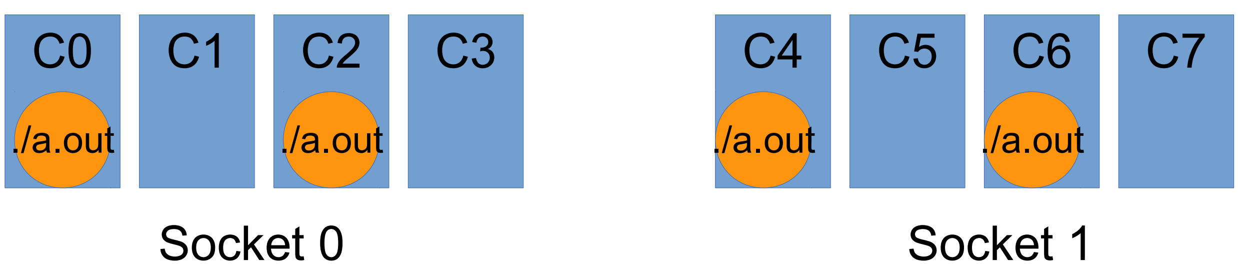
taskset

Bitmask is 1 where the process should be allowed to run – in this case 00111111 = 0x3f

- `$ taskset 0x3f ./a.out`



- `$ taskset -p 0,2,4,6 ./a.out`



Thread placement - numactl

- numactl is a command of the operating system providing similar functionality as taskset, but very much focused on the NUMA features of a system. numactl understands which processors form a NUMA node and how threads need to be grouped together

Thread placement - numactl

chris@snb85:~

```
[chris@snb85 ~]$ numactl --show
policy: default
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
cpubind: 0 1
nodebind: 0 1
membind: 0 1
[chris@snb85 ~]$
```

chris@snb85:~

```
[chris@snb85 ~]$ numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 16 17 18 19 20 21 22 23
node 0 size: 32691 MB
node 0 free: 31518 MB
node 1 cpus: 8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31
node 1 size: 32768 MB
node 1 free: 31932 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
[chris@snb85 ~]$
```

Thread placement - numactl

<u>Numactl</u>

--membind <n>: place pages on NUMA node <n>

--cpunodebind <n>: pin threads to node <n>

--interleave <nodes>: put the pages round-robin on <nodes>

<u>Example:</u>

numactl --cpunodebind=0 --membind=0,1 ./a.out

This puts memory on nodes 0 an 1, but threads only on node 0.

Thread placement - OpenMP Affinity

- Both numactl and taskset allow you to set an affinity of a set of threads, but not the affinity of a thread within the set. The OS will still schedule threads from one core to another (even if not from a NUMA node to the next)
- Specifying OpenMP affinity environment variables allows the detailed control of individual thread placement.

Thread placement - OpenMP Affinity (Intel)

- Affinity of threads for OpenMP binaries compiled with the Intel compiler are controlled over the KMP_AFFINITY environment variable

- KMP_AFFINITY=[<modifier>,...]<type>[,<permute>][,<offset>]

modifier
granularity=<specifier>
specifiers: fine, thread, and **core**
norespect
noverbose
nowarnings
proclist={<proc-list>}
respect
verbose
warnings

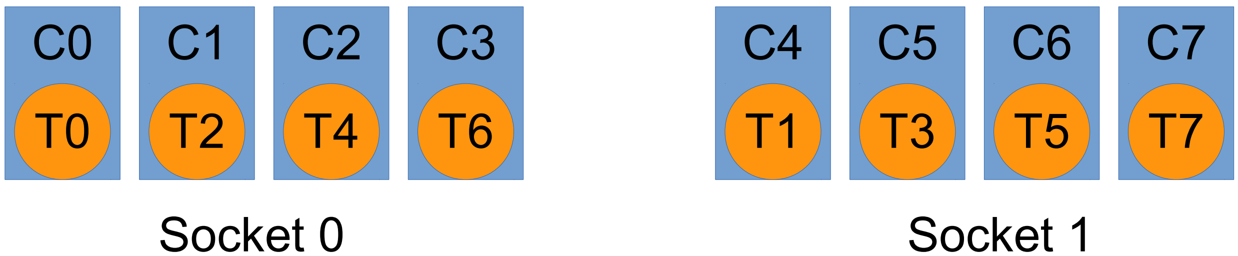
type
compact
disabled
explicit
none
scatter

permute and offset
Both are integers
0

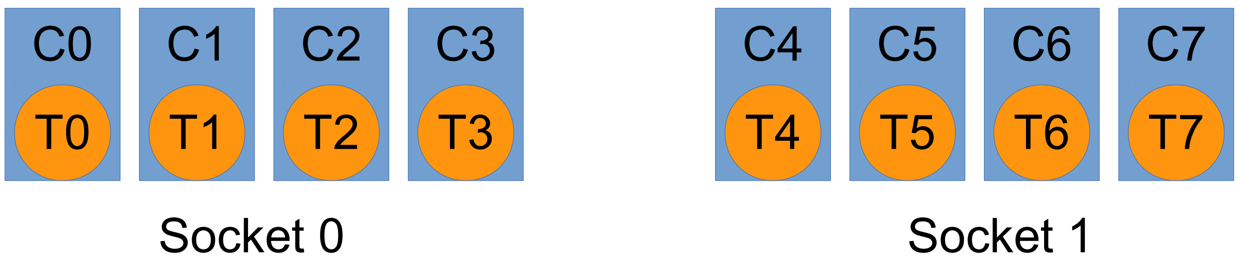
Defaults are **red**

Thread placement - OpenMP Affinity (Intel) Simple usage (no HT)

- KMP_AFFINITY=scatter

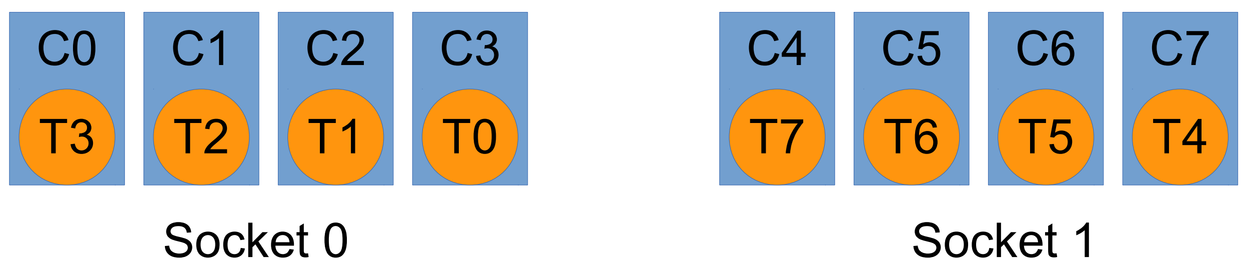


- KMP_AFFINITY=compact

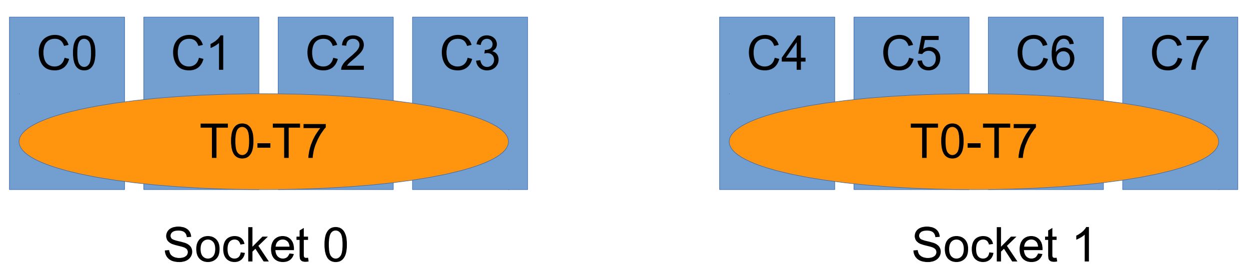


Thread placement - OpenMP Affinity (Intel)

- KMP_AFFINITY=explicit,proclist=[3,2,1,0,7,6,5,4]



- KMP_AFFINITY=none



Summary

- In (ubiquitous) NUMA systems, proper thread and process placement is a must
- Numctl and taskset are OS tools to do so
- KMP_AFFINITY is a way to easily place OpenMP threads with the Intel compiler