MHPC
Master in High Performance Computing

Lecture 5
Multicore architecture and how to use them at best

Stefano Cozzini

CNR/IOM and eXact-lab srl

SISSA
Scuola Internazionale Superiore di Studi Avanzati

ICTP The Abdus Salam International Centre for Theoretical Physics

MHPC
Master in High Performance Computing

Agenda

- Introduction: again why multicore ?
- Architectures of multicore/ Issues in using Multicore architectures
- Threads placement on multicore/multiprocessor machine
  - Hwloc
  - Numactl
  - OpenMP approach
- Evaluate performance of multicore node (tutorial/exercises)
  - Stream to measure memory
  - MPI benchmark to measure latency among different cores
  - Nodeperf code

SISSA ICTP

MHPC
Master in High Performance Computing

Goal of the day

- Get acquainted of basic brick on modern HPC system
- Learn about pro/cons of such architecture
- Start using tools to correctly exploit (almost) all the cores of the architecture

SISSA ICTP

MHPC
Master in High Performance Computing

RECAP

- Modern nodes are multiprocessor (more than one CPUs) and each CPU is multicore
- RAM Memory is shared among all cores
- L1/L2 caches are private to cores
- L3 is shared within the same CPU
- The overall architecture is NUMA

SISSA ICTP

MHPC
Master in High Performance Computing

Motivation for multicores

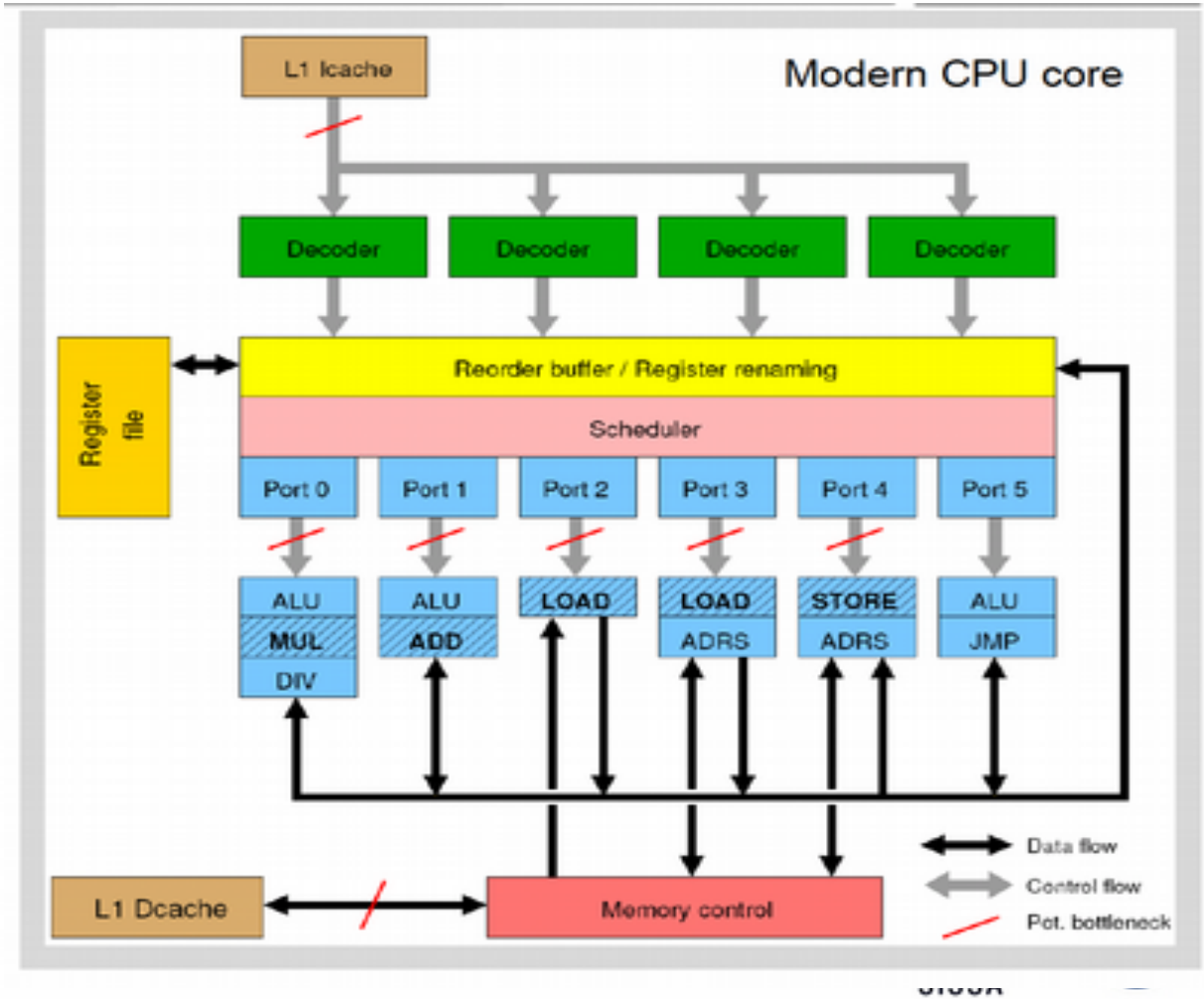
- Exploits increased feature-size and density
- Increases functional units per chip (spatial efficiency)
- Limits energy consumption per operations
- Constrains growth in processor complexity

SISSA ICTP

MHPC
Master in High Performance Computing

Recap: how does a core contain ?

- Turing machine
- Similar design on all modern systems



MHPC
Master in High Performance Computing

Recap: elements in a core

- Pipelining of arithmetic/functional units
  - Split complex instruction into several simple / fast steps (stages)
  - Execute different steps on instructions at the same time (in parallel)
  - after the pipeline is full one result at each cycle
- Superscalar processor:
  - Multiple units enable Istruction Level Parallelism (ILP)
  - 2-6 way superscalar : 2-6 istructions at each cycle
- SIMD extensions (AVX/SSE2)
  - Single Instruction Multiple data approach: 2/4/8 FP at each cycle
  - Loop unrolling techniques...
  - No SIMD approach on loop with dependencies: A(I)= A(I-1) *B(I)
  - Pointer aliasing may prevent SIMD...

SISSA ICTP

Recap: Peak performance on a core

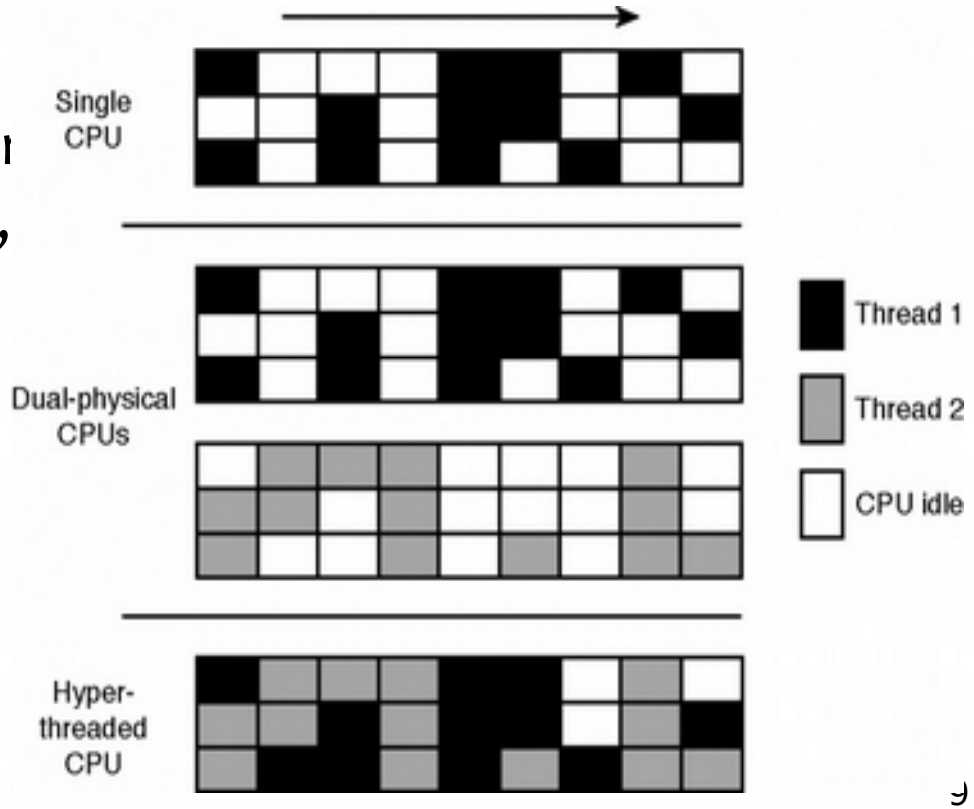
$$P_{core} = n_{super}^{FP} \cdot n_{FMA} \cdot n_{SIMD} \cdot f$$

Super-scalarity | FMA factor | SIMD factor | Clock Speed

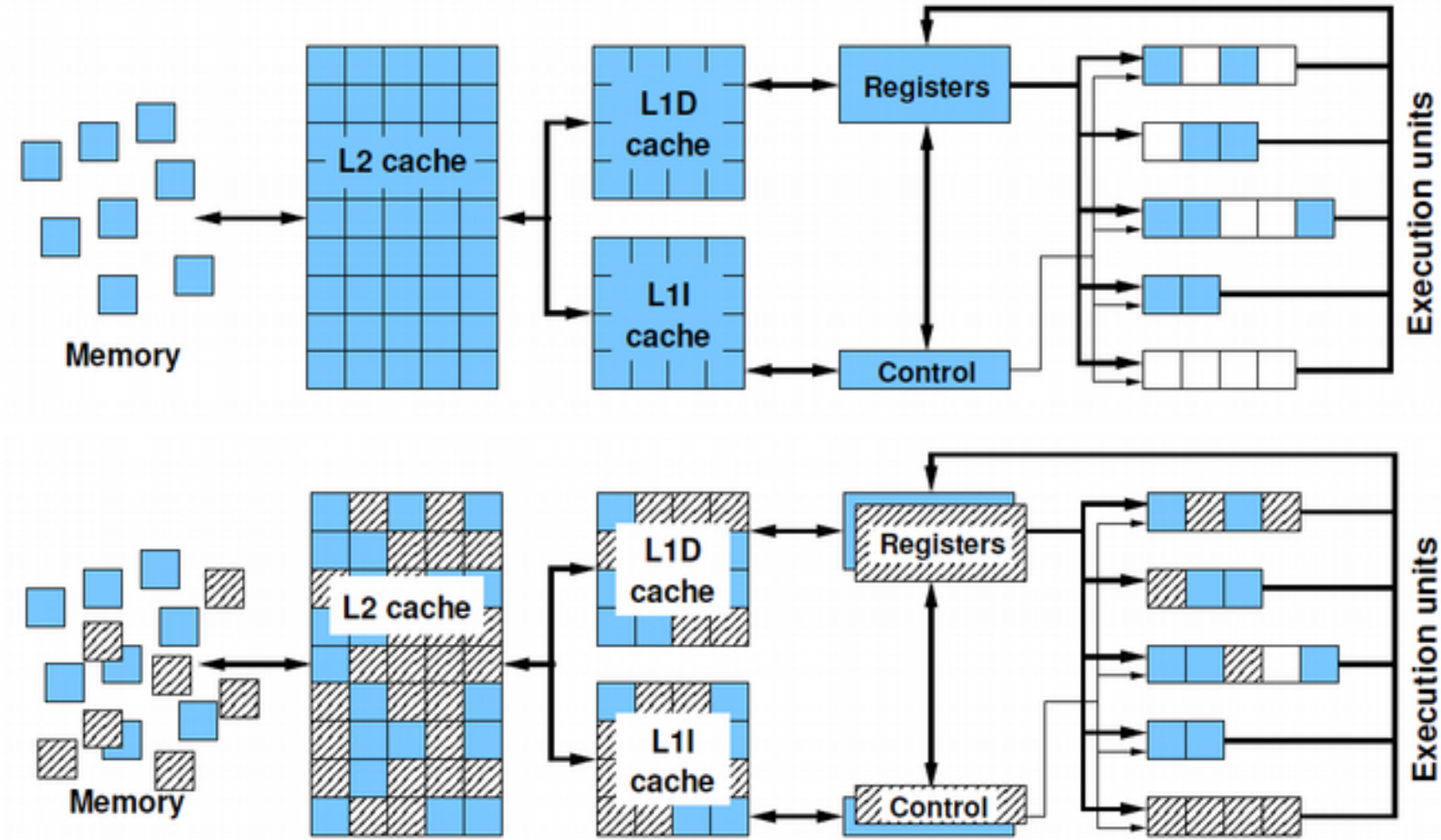
Typical representatives	n_{super}^{FP} inst./cy	n_{FMA}	n_{SIMD} ops/inst.		Code	f [GHz]	P_{core} [GF/s]
Nehalem	2	1	2	Q1/2009	X5570	2.93	11.7
Westmere	2	1	2	Q1/2010	X5650	2.66	10.6
Sandy Bridge	2	1	4	Q1/2012	E5-2680	2.7	21.6
Ivy Bridge	2	1	4	Q3/2013	E5-2660 v2	2.2	17.6
Haswell	2	2	4	Q3/2014	E5-2695 v3	2.3	36.8
Broadwell	2	2	4	Q1/2016	E5-2699 v4	2.2	35.2

Hyper threading (HT)

- Intel® Hyper-Threading Technology uses processor resources more efficiently, enabling multiple threads to run on each core.
- O.S. "sees" two cores and transparently try to execute two program on two different "cores"
- Generally bad for HPC ?



SMT principle (2 way)



MHPC
Master in High Performance Computing

Does SMT/HT work for HPC kernel?

- SMT/HT introduce an additional topological layer inside the core
- All caches and PU are shared !
- Possible advantage: better pipeline throughput
  - Filling otherwise unused pipeline
  - Filling pipelines bubbles with other threads execution's instructions

Thread 0:

```
do i=1,N
  a(i) = a(i-1)*c
enddo
```

Dependency → pipeline stalls until previous MULT is over

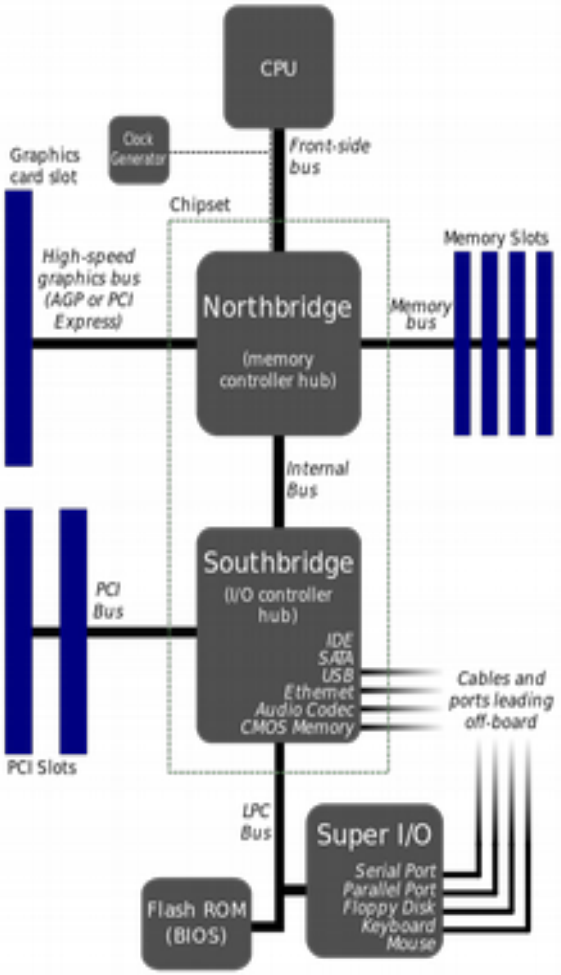
Thread 1:

```
do i=1,N
  b(i) = s*b(i-2)+d
enddo
```

Unrelated work in other thread can fill the pipeline bubbles

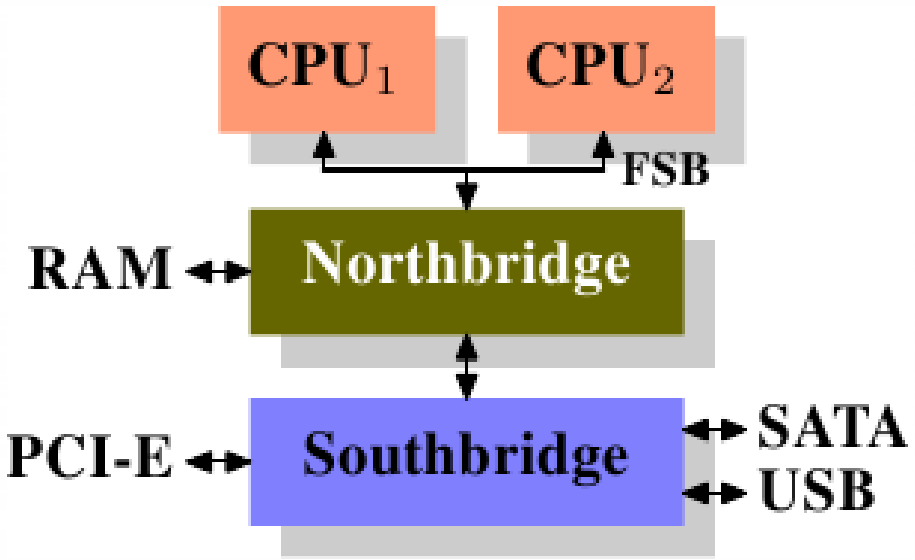
standard modern architecture

- All data communication from one CPU to another must travel over the same bus used to communicate with the Northbridge.
- All communication with RAM must pass through the Northbridge.
- Communication between a CPU and a device attached to the Southbridge is routed through the Northbridge.



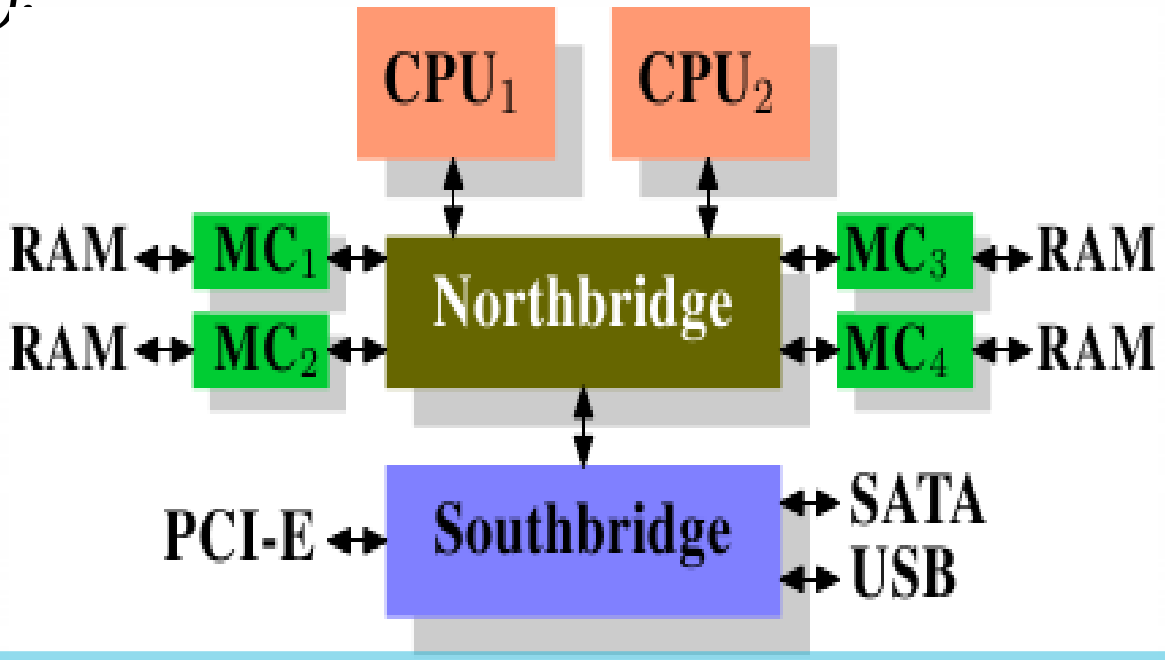
standard multisocket architecture

- Characteristics:
  - more than one CPU !
  - 64 bit address space



more expensive and modern architecture

- Northbridge can be connected to a number of external memory controllers (in the following example, four of them).

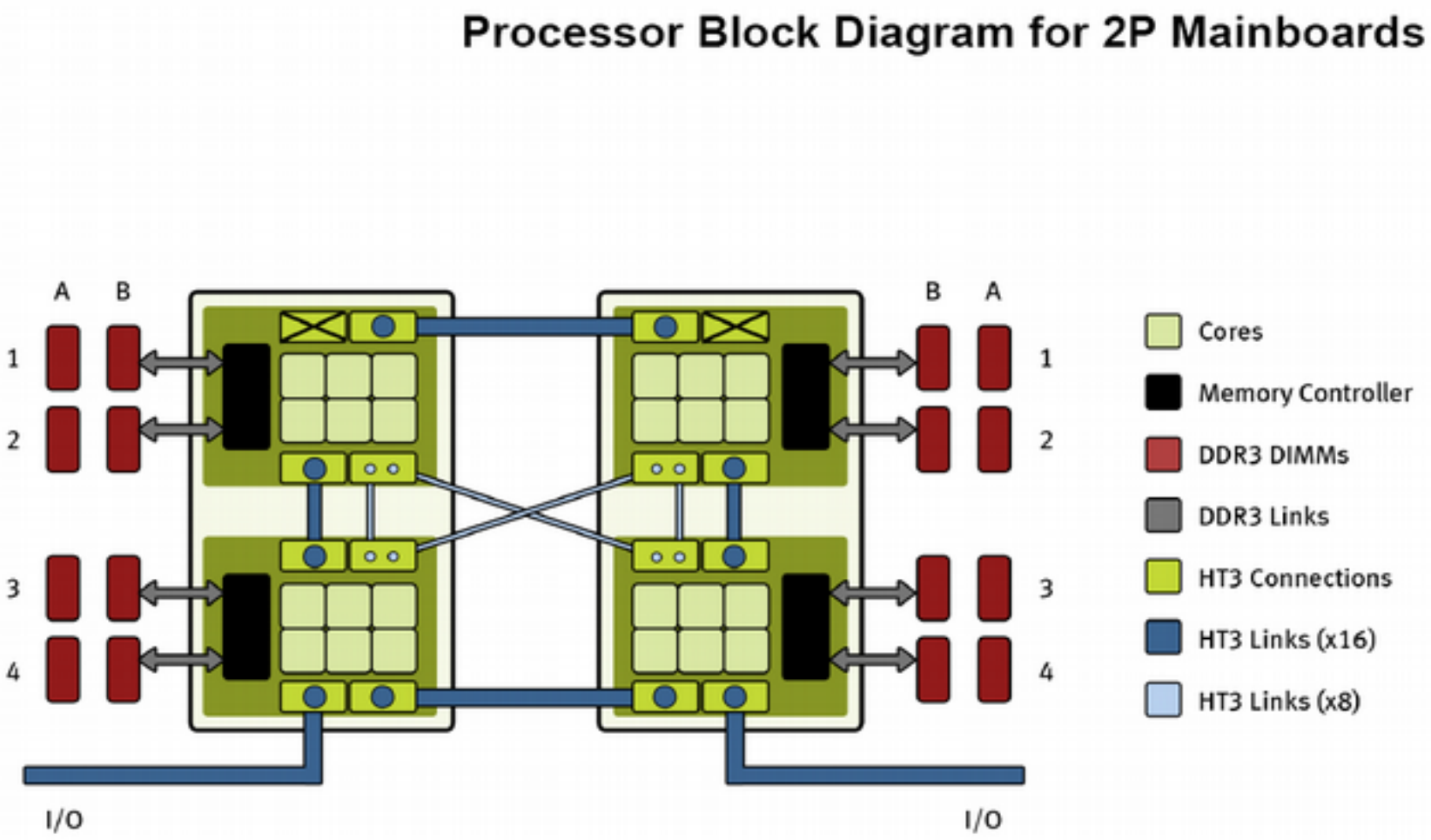


INCREASE IN BANDWIDTH TOWARD MEMORY

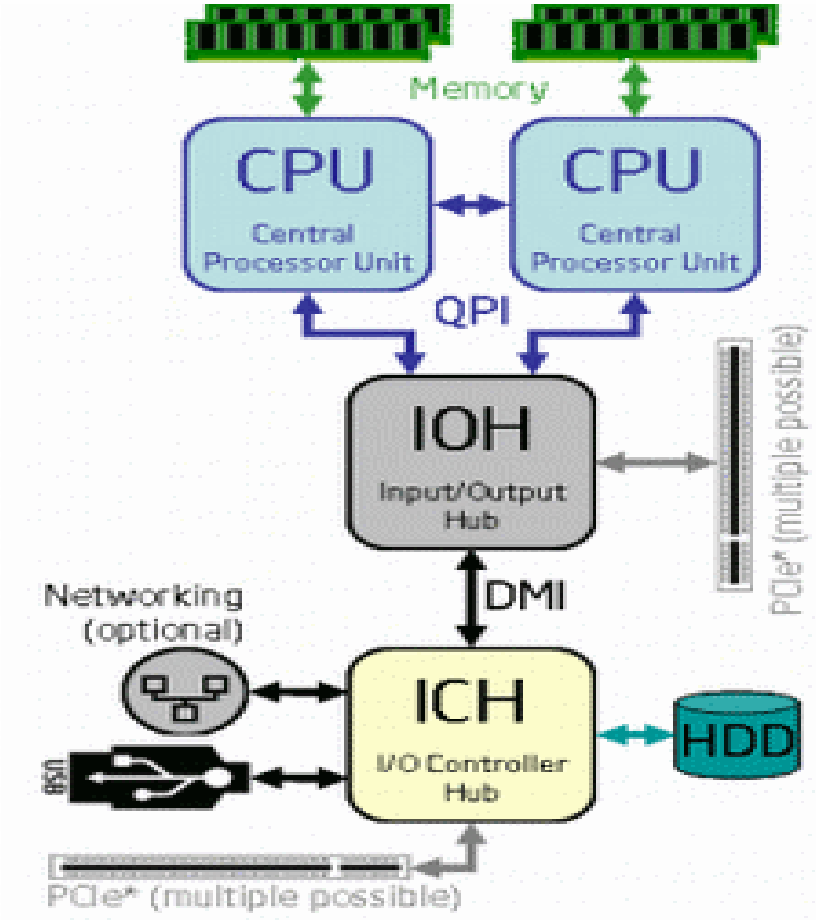
From SMP to NUMA

- FSB became rapidly a bottleneck: all the CPUs accessing memory throught it
- SMP (UMA) approach no longer possibile
- First NUMA architecture:
  - Hypertransport technology by AMD (2005)
- Intel came much later
  - Quick Path Interconnect (2009)

Opteron 6xxx AMD CPU



Xeon Family: Nehalem introduces NUMA



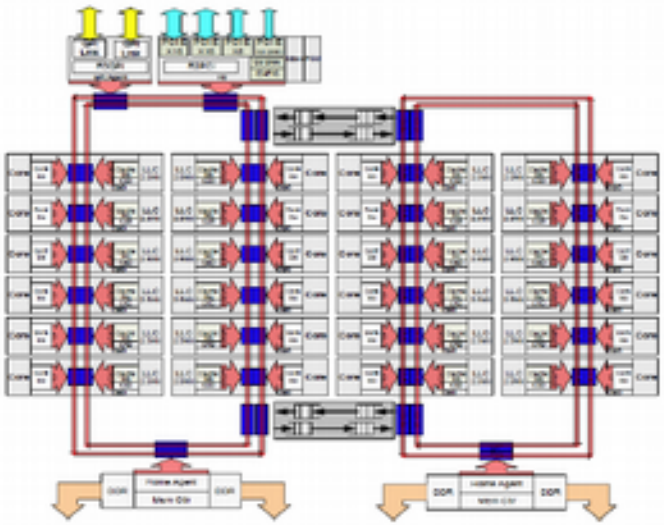
- First NUMA architecture by INTEL
- QPI among CPUs to play the role of hyper-transport in AMD
- Released April 2009

MHPC
Master in High Performance Computing

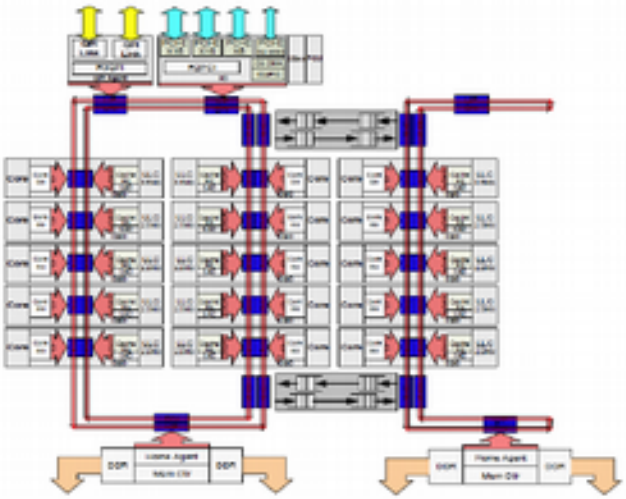
Brodwell layout

Broadwell EP die configurations

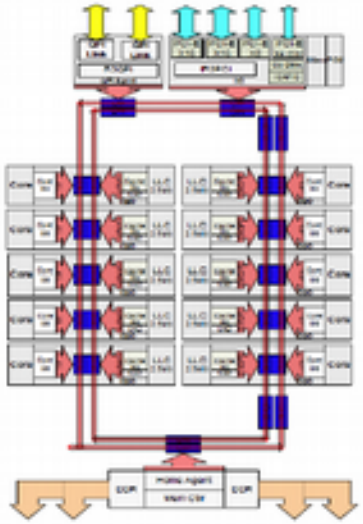
Up to 22 Core (HCC)



Up to 14 Core (MCC)



Up to 8 Core (LCC)



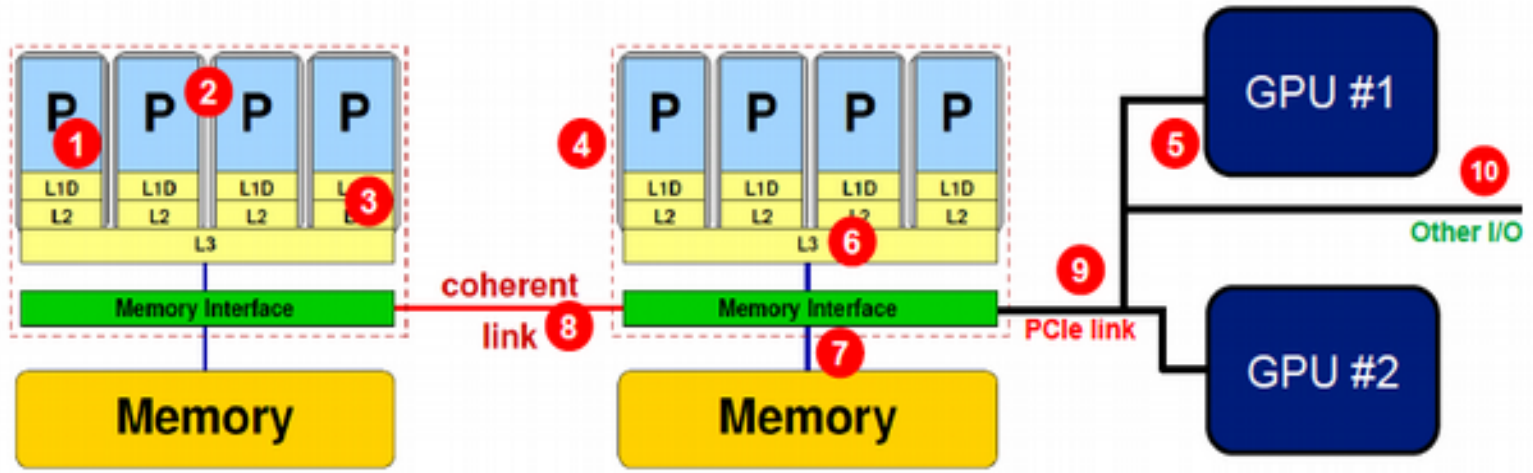
Chop	Columns	Home Agents	Cores	Power (W)
HCC	4	2	12-22	105-145
MCC	3	2	8-14	85-120
LCC	2	1	6-8	85

Challenges for multicore

- Relies on effective exploitation of multiple-thread parallelism
  - Need for parallel computing model and parallel programming model
- Aggravates memory wall problem
  - Memory bandwidth
    - Way to get data out of memory banks
    - Way to get data into multi-core processor array
    - Memory latency
  - Cache sharing

MHPC
Master in High Performance Computing

Parallellism and shared resource on shared-memory node

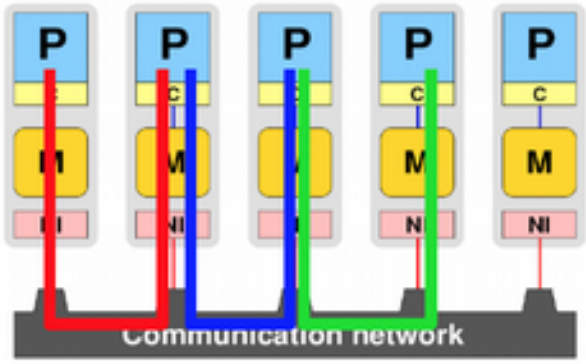


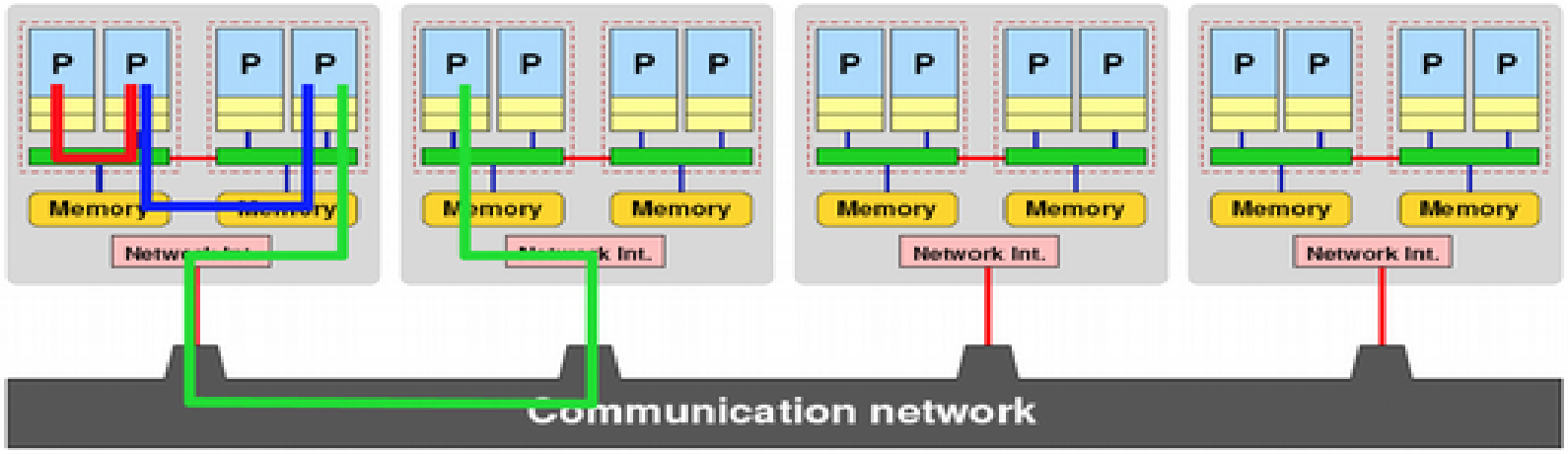
- Parallel resources
  - Execution/SIMD units (1)
  - Cores (2)
  - Inner cache levels (3)
  - Socket/ccNuma domains (4)
  - Multiple accelerator (5)
- Shared resources
  - Outer cache level per socket (6)
  - Memory bus per socket(7)
  - Intersocket link (8)
  - PCI-bus(es) (9)
  - Other I/O resources (10)

SISSA ICTP

Parallel programming model : MPI

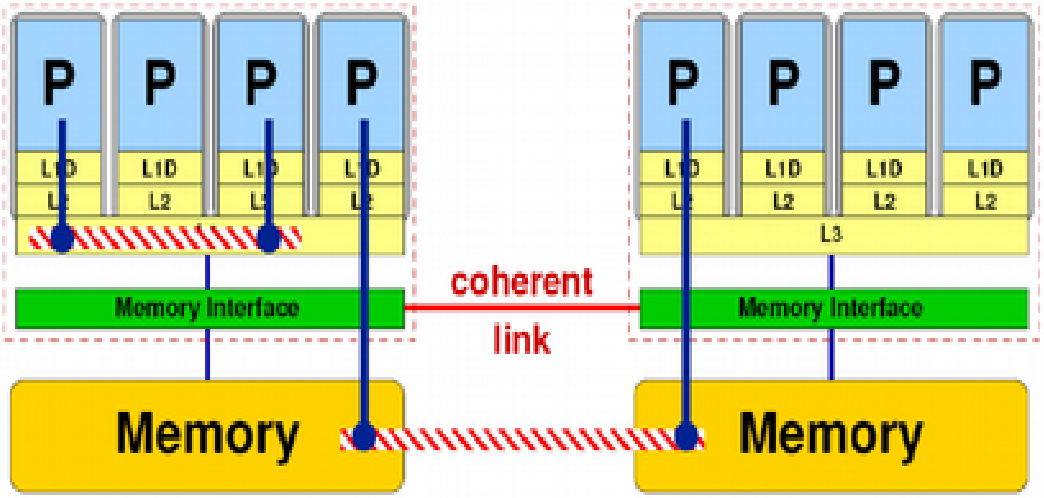
- Machine structure is invisible to user
  - Very simple programming model
  - MPI "knows what to do"!?
- Performance issues
  - Intranode vs. internode MPI
  - Node/system topology

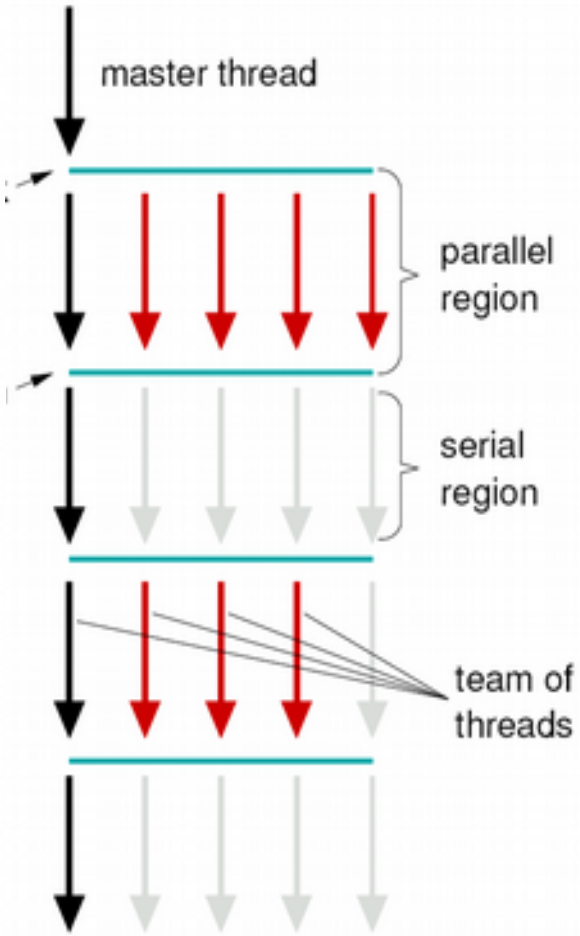


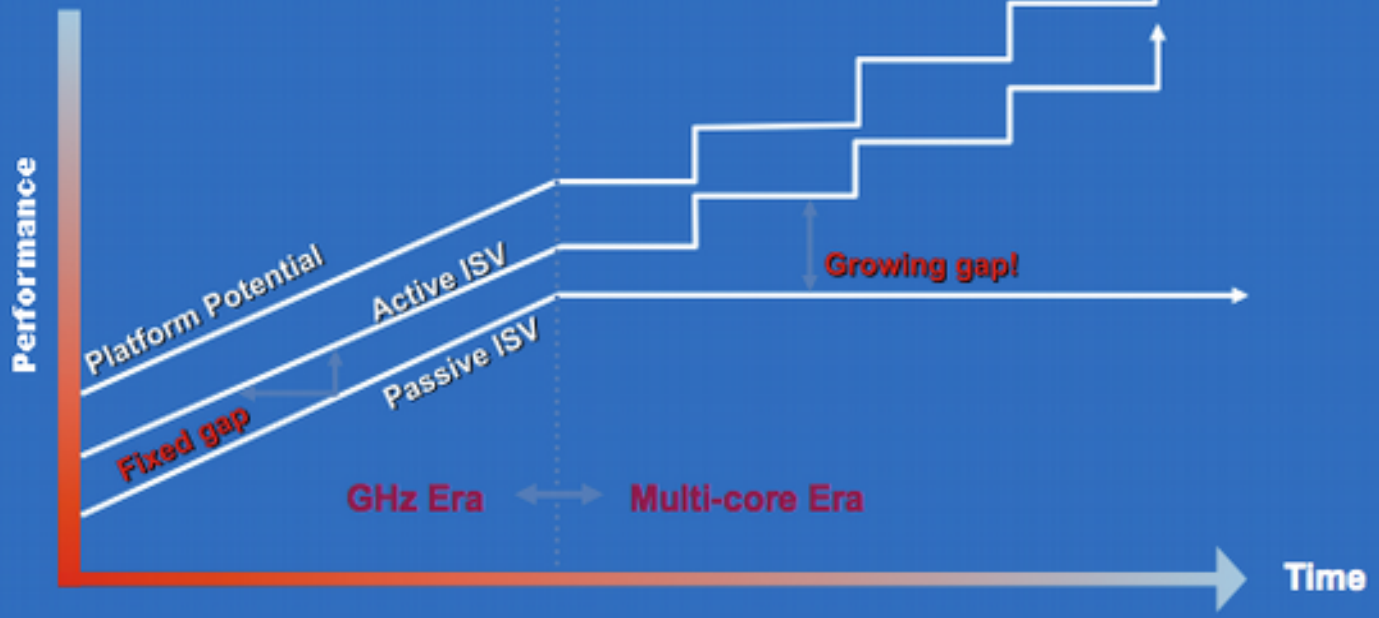


Parallel programming model : multithreaded

- Machine structure is invisible to user
  - Very simple programming model
  - Threading SW (OpenMP, pthreads, TBB,…) should know about the details
- Performance issues
  - Synchronization overhead







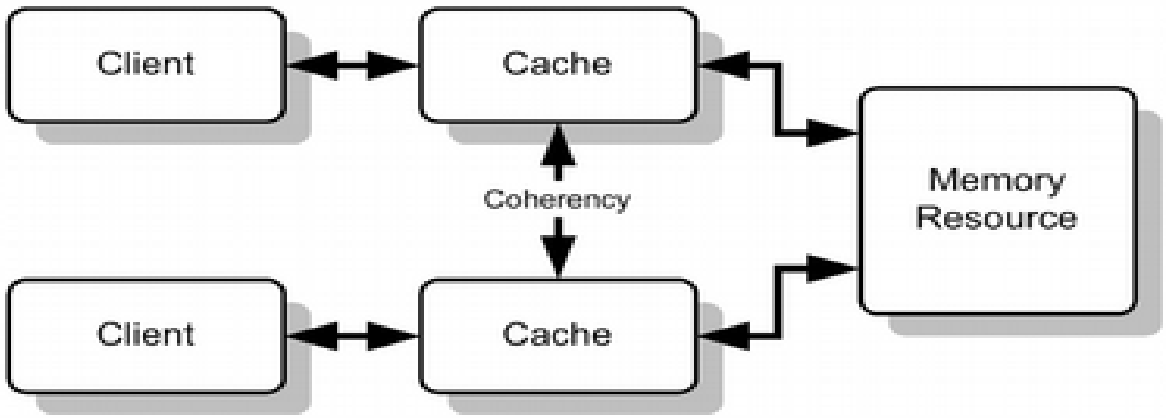
"Parallelism for Everyone"

Parallelism changes the game

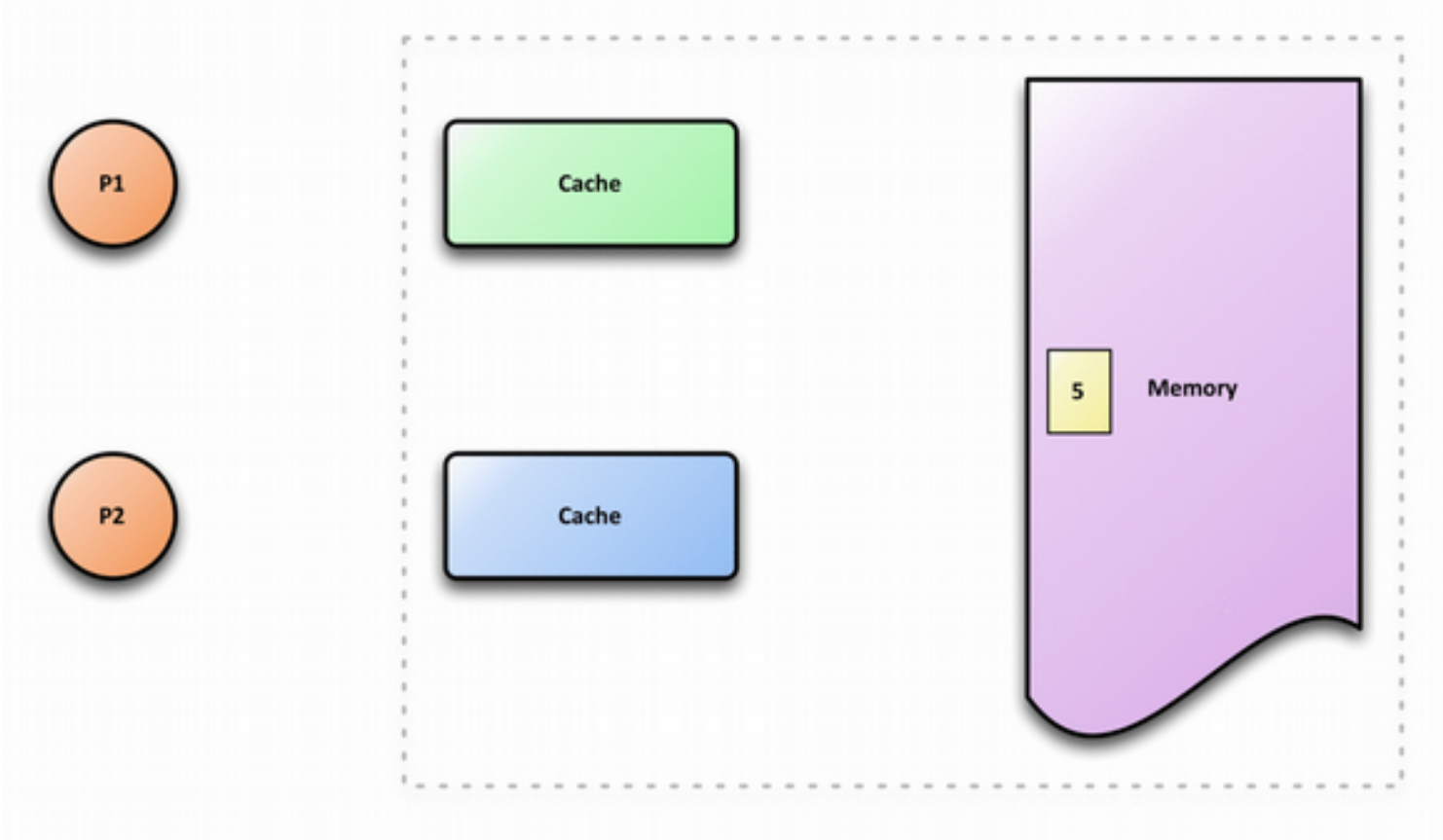
- A large percentage of people who provide applications are going to have to care about parallelism in order to match the capabilities of their competitors.

Cache coherency

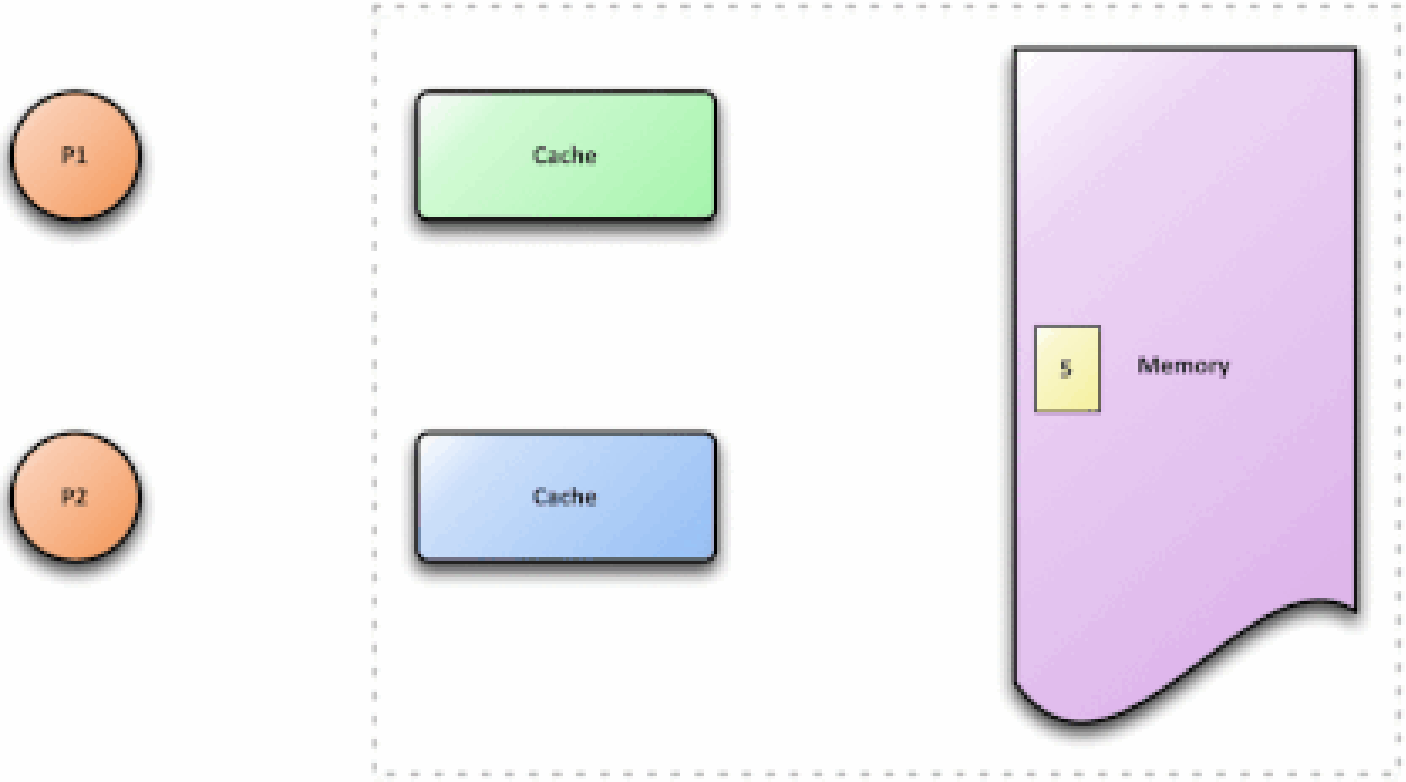
- From wikipedia:
  - the uniformity of shared resource data that ends up stored in multiple local caches. When clients in a system maintain caches of a common memory resource, problems may arise with incoherent data, which is particularly the case with CPUs in a multiprocessing system.



Incoherent caches (from animated gif in wikipedia)



Coherent caches (from animated gif in wikipedia)



MHPC
Master in High Performance Computing

Again on cache coherency: false sharing problem

- Consider the following example:

```
for (i=0; i<10; i++)
        a[i]= b[i] + c[i];
```

- Let us assume we run this on 2 processors:

processor 1 for I=0,2,4,6,8

processor 2 for I=1,3,5,7,9

SISSA ICTP

MHPC
Master in High Performance Computing

Let us use it..

lstopo – Displaying topology information

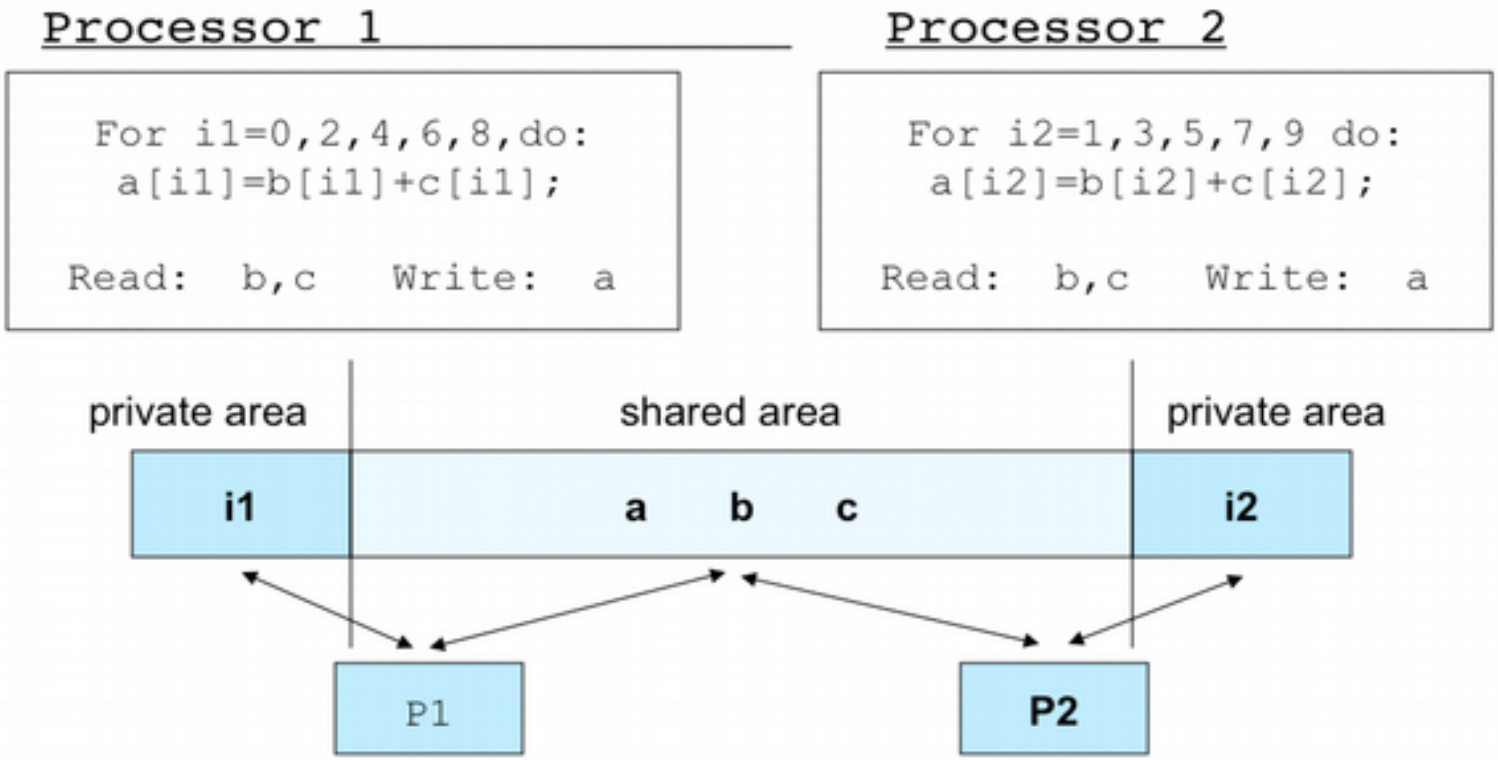
hwloc-distances – show object distances

Notably NUMA distances:

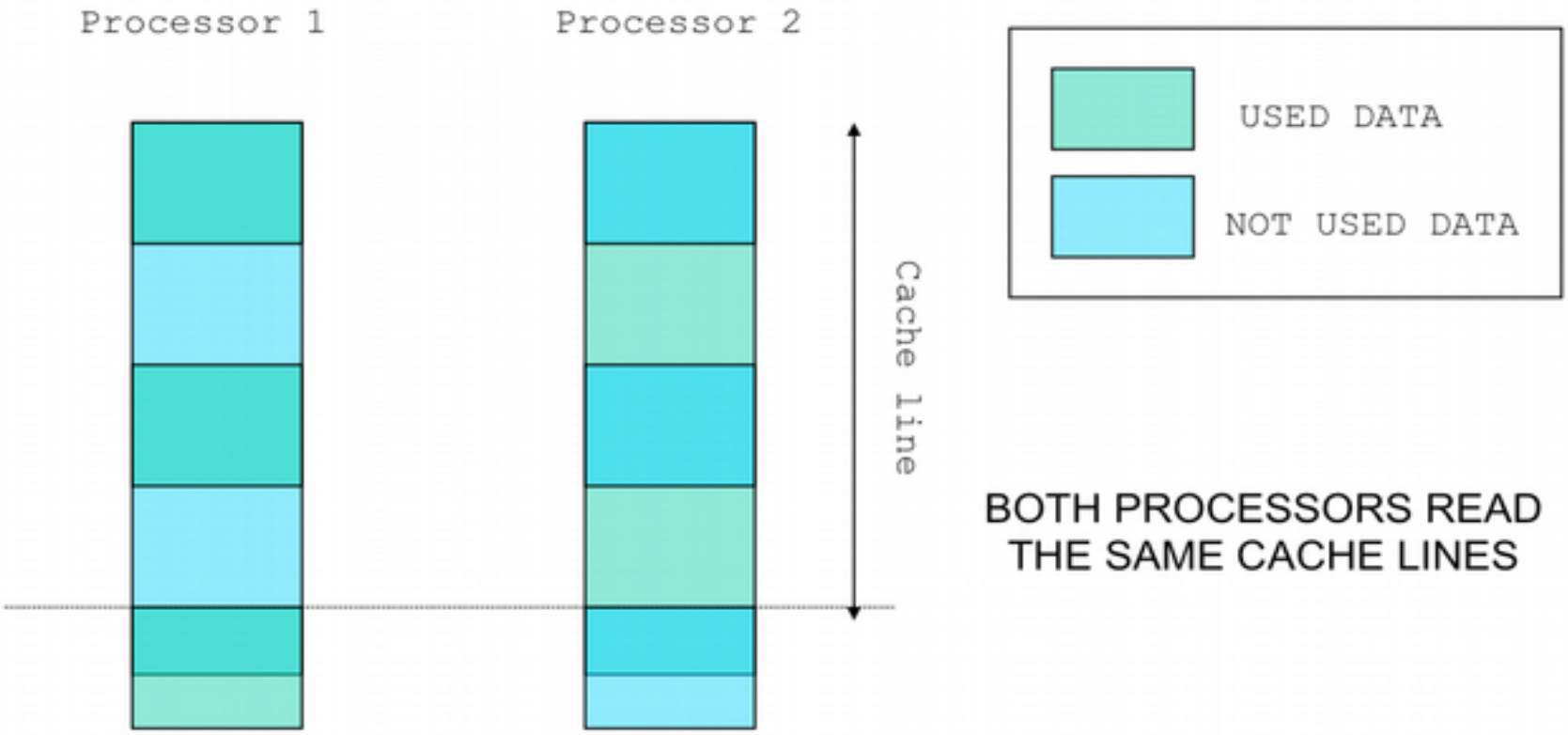
```
$ ./utils/hwloc-distances
```

SISSA ICTP

What is happening ? (1)



What is happening ? (2)



Shared memory - NUMA

- Dual socket nodes can operate in two modes

NUMA
(numa=on)

Socket 0

0
1
2
3
4
5
6
7

Socket 1

8
9
10
11
12
13
14
15

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

0
1
2
3

Interlaced
(numa=off)

Socket 0

0
2
4
6
8
10
12
14

Socket 1

1
3
5
7
9
11
13
15

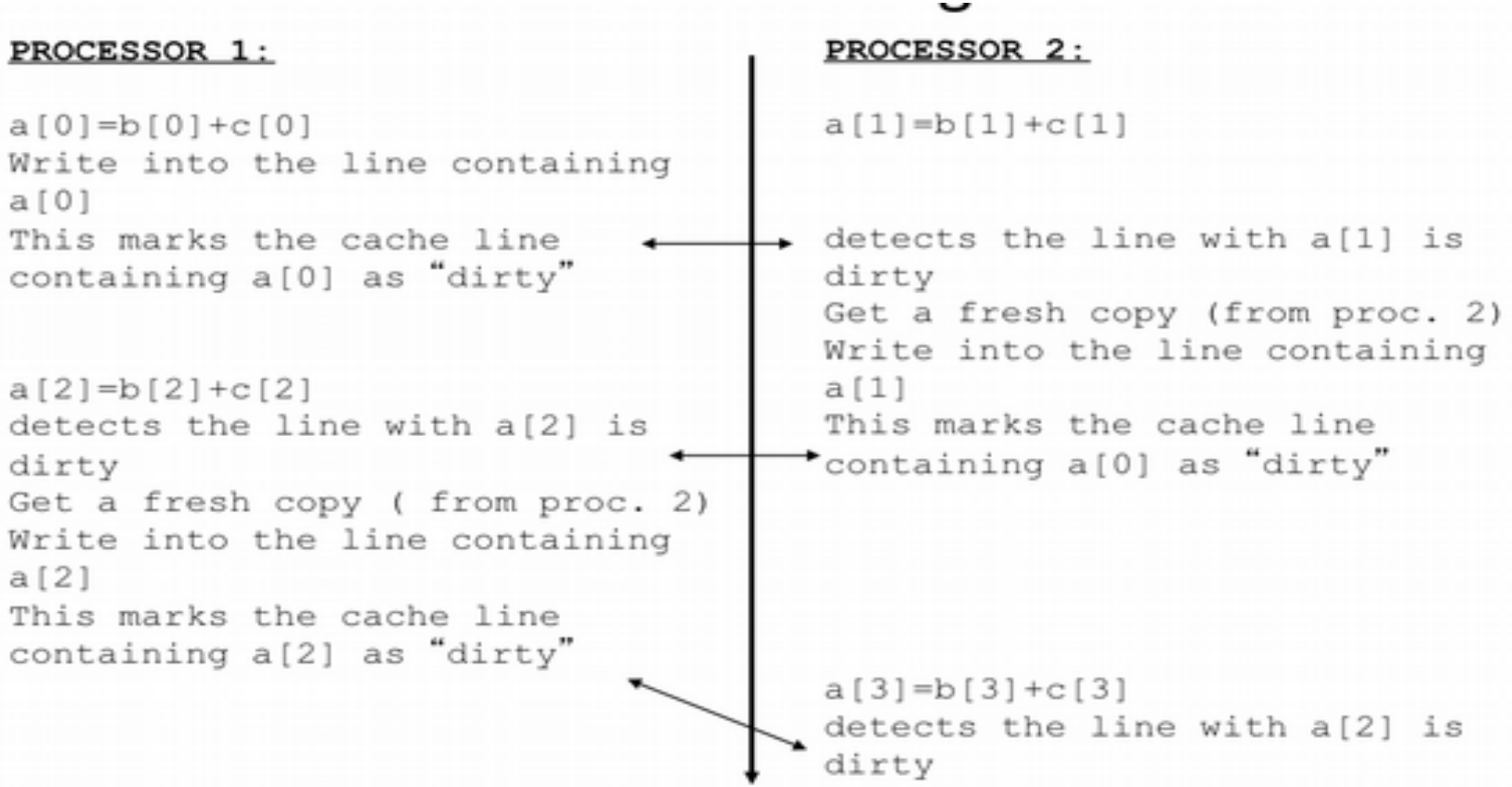
Memory controller maps each other cache-line from another socket to contiguous memory. Allocated memory might look like this:

0
1
2
3

NUMA mode generally gives the best results for HPC

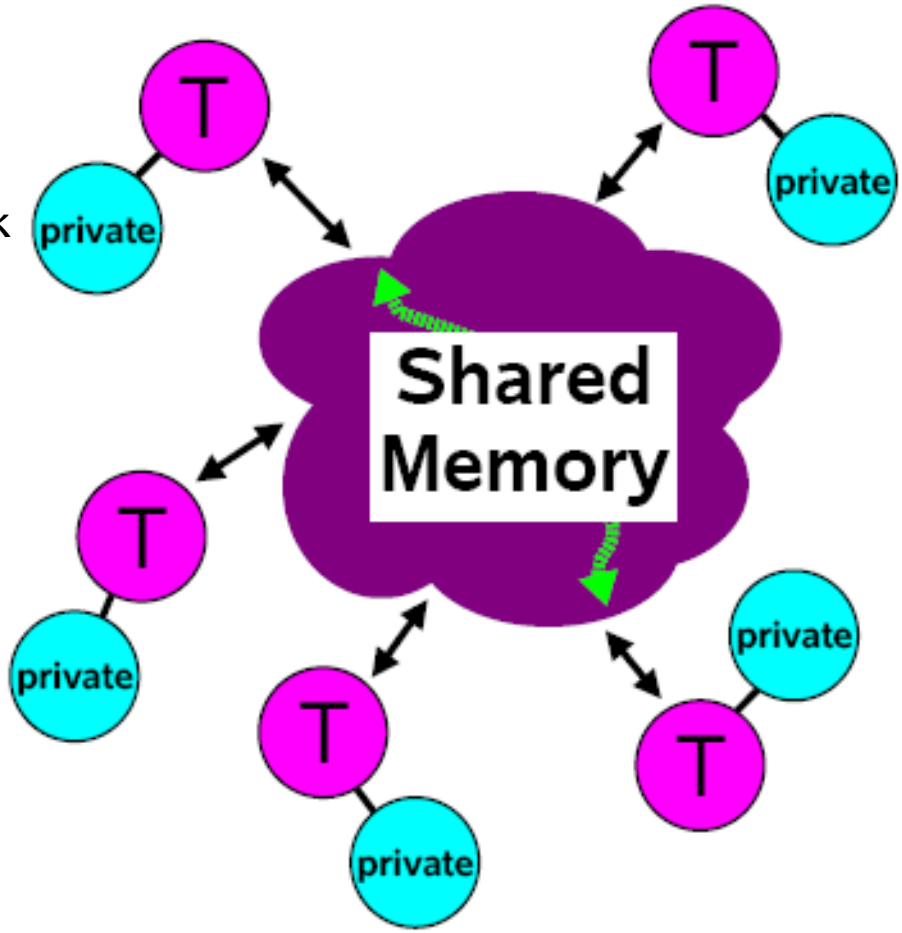
if you are careful, you will always get the fast local memory performance

What is happening ? (3)



the shared programming model

- it assumes global data and explicit creation of execution threads that work on that data
- All threads have access to the same, globally shared,memory
- Data can be shared or private
  - Shared data is accessible by all threads
  - Private data can be accessed only by the threads that owns it
- Data transfer is transparent to the programmer
- Synchronization takes place,but it is mostly implicit



MHPC
Master in High Performance Computing

hello world in openMP

```
#include <omp.h>
int main()  {
  int iam =0, np = 1;
#pragma omp parallel private(iam, np)
  {
#if defined (_OPENMP)
   np = omp_get_num_threads();
   iam = omp_get_thread_num();
#endif
  printf("Hello from thread %d out of %d \n", iam, np);
 }
}
```

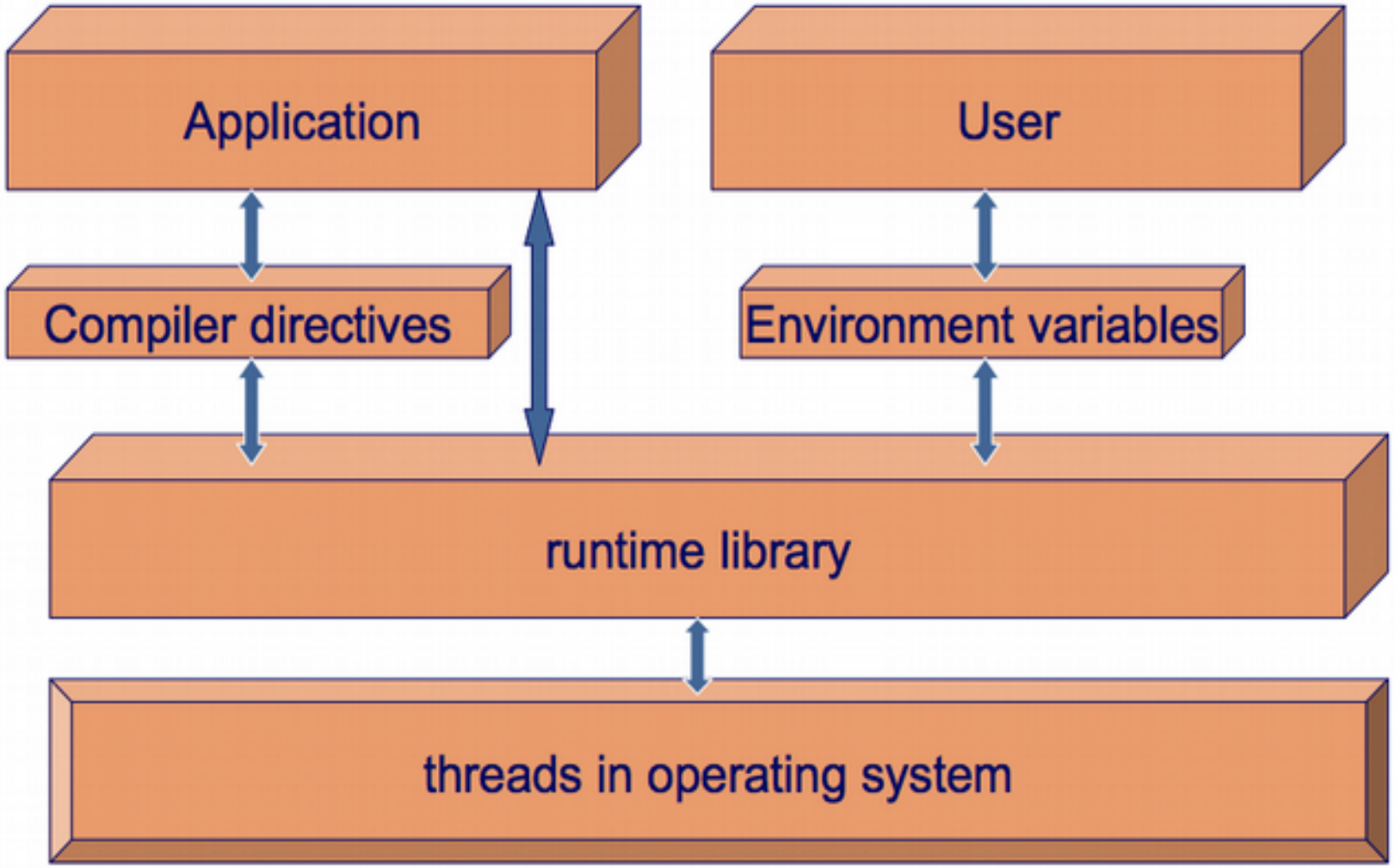

SISSA ICTP

compile&run the code (tutorial)

- Gnu:

```
gcc -fopenmp omp_101.c  -o omp_101.x
./omp_101.x
```

HOW MANY THREADS DID YOU GET ?


Application
User
Compiler directives
Environment variables
runtime library
threads in operating system

MHPC
Master in High Performance Computing

compile&run the code (tutorial)

```
[exact@master openmp101]$ gcc omp_101.c
[exact@master openmp101]$ ldd a.out
   linux-vdso.so.1 =>  (0x00007fffae5ff000)
   libc.so.6 => /lib64/libc.so.6 (0x00000038a1a00000)
   /lib64/ld-linux-x86-64.so.2 (0x00000038a1600000)

[exact@master openmp101]$ gcc -fopenmp omp_101.c
[exact@master openmp101]$ ldd a.out
   linux-vdso.so.1 =>  (0x00007fff511ff000)
   libgomp.so.1 => /usr/lib64/libgomp.so.1 (0x00000038ec60000)
   libpthread.so.0 => /lib64/libpthread.so.0 (0x00000038a2200)
   libc.so.6 => /lib64/libc.so.6 (0x00000038a1a00000)
   librt.so.1 => /lib64/librt.so.1 (0x00000038a2e00000)
   /lib64/ld-linux-x86-64.so.2 (0x00000038a1600000)
```


SISSA ICTP

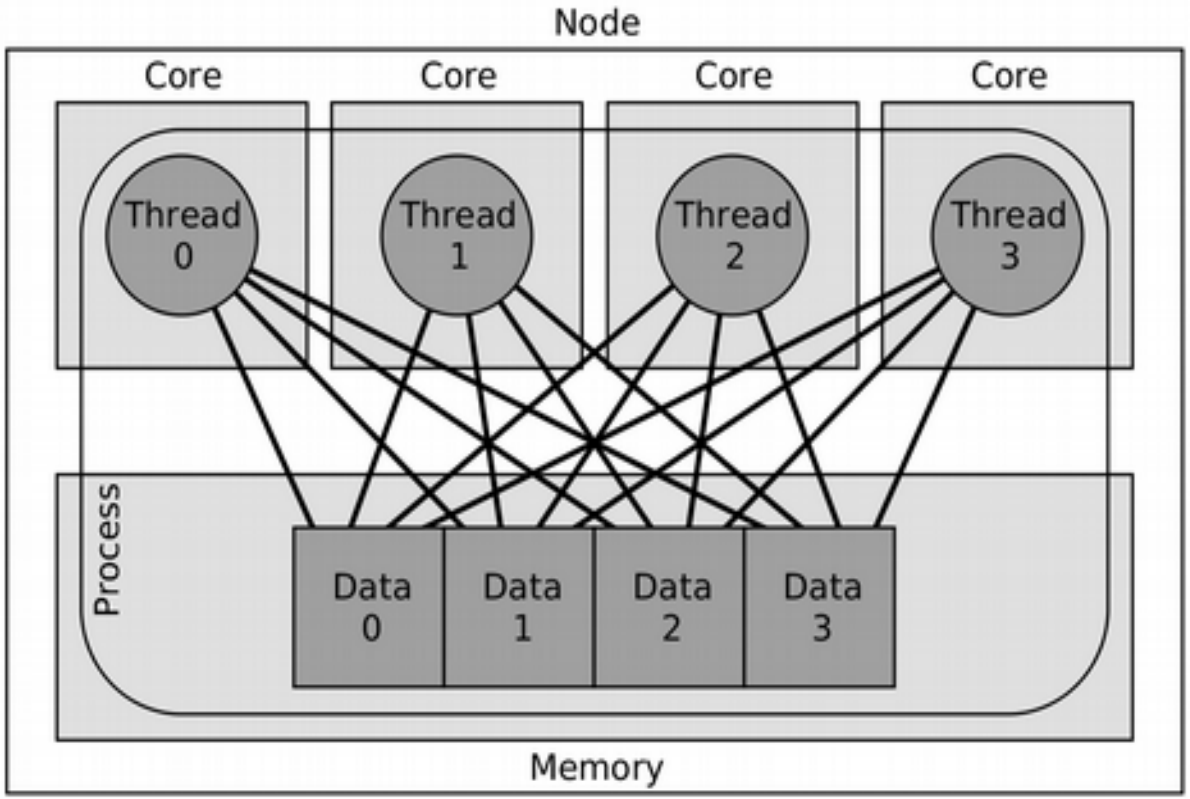
Interacting with users (tutorial)

- Define environment variable

```
[exact@master openmp101]$ export OMP_NUM_THREADS=2
[exact@master openmp101]$ ./a.out
Hello from thread 0 out of 2
Hello from thread 1 out of 2

[exact@master openmp101]$ export OMP_NUM_THREADS=4
[exact@master openmp101]$ ./a.out
Hello from thread 2 out of 4
Hello from thread 0 out of 4
Hello from thread 3 out of 4
Hello from thread 1 out of 4
```

Threads and memory placement



Many important reason to care about threads placement

- Eliminating performance variation
- Making use at best of architectural features
- Avoiding resource contention

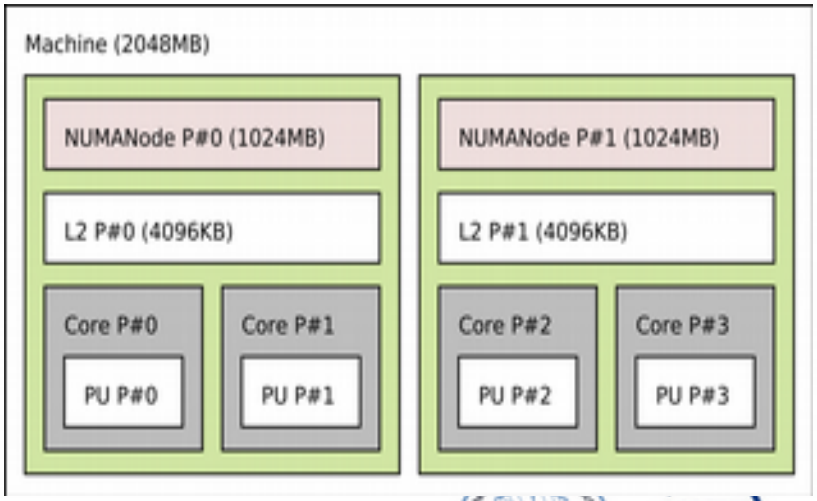
MHPC
Master in High Performance Computing

Our dilemma

- Use cores 0 & 1 to share cache and improve synchronization cost?
- Use cores 0 & 2 to maximize memory bandwidth?
- How to choose portability?

Depends on

- the application structure
- machine structure



SISSA ICTP

Thread Placement

How to keep threads on a particular core, so data is readily available when needed ?

- Many possible ways:
  - Numactl
  - Likwid-pin
  - hwloc
  - OpenMP (OMP_PLACES/ OMP_PROC_BIND/KMP_AFFINITY)

MHPC
Master in High Performance Computing

Thread placement - numactl

- numactl is a command of the operating system providing 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

SISSA ICTP

Thread placement - numactl

```
[cozzini@cn02-03 ~]$ 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
cpubind: 0 1
nodebind: 0 1
membind: 0 1
```

```
[cozzini@cn02-03 ~]$ numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9
node 0 size: 20451 MB
node 0 free: 18675 MB
node 1 cpus: 10 11 12 13 14 15 16 17 18 19
node 1 size: 20480 MB
node 1 free: 19538 MB
node distances:
node   0   1
  0:  10  11
  1:  11  10
```

MHPC
Master in High Performance Computing

Thread placement - numactl

Numactl

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

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

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

Example:

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

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

SISSA ICTP

MHPC
Master in High Performance Computing

Thread placement - OpenMP Affinity

- numactl allows 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.

SISSA ICTP

MHPC
Master in High Performance Computing

Thread placement in OpenMP

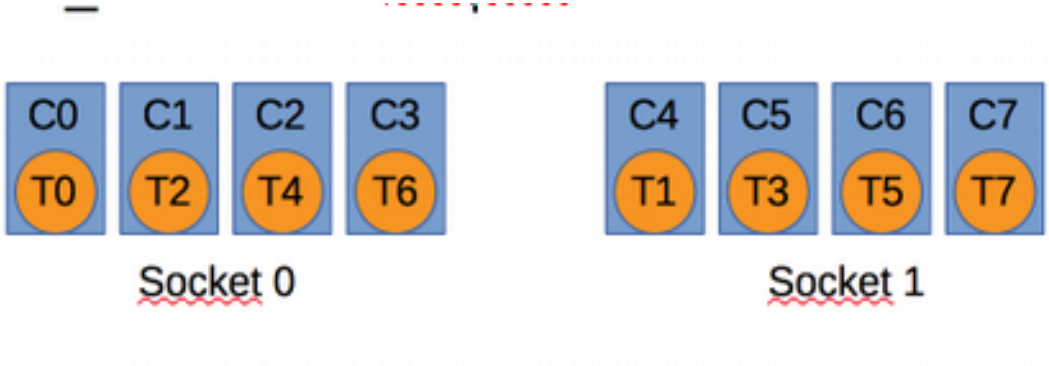
- Thread placement can be controlled with two environment variables:
- the environment variable OMP_PROC_BIN describes how threads are bound to OpenMP places
- the variable OMP_PLACES describes these places in terms of the available hardware.
- When you're experimenting with these variables it is a good idea to set OMP_DISPLAY_ENV to true, so that OpenMP will print out at runtime how it has interpreted your specification.

SISSA ICTP

Some examples:

```
Export OMP_PLACES=sockets
```

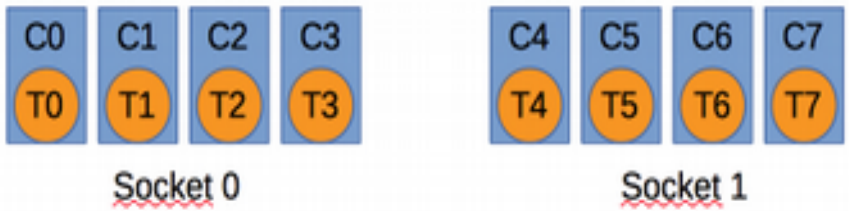
- Then
  - thread 0 goes to socket 0,
  - thread 1 goes to socket 1,
  - thread 2 goes to socket 0 again,
  - and so on.



Some examples

```
export OMP_PLACES=cores
export OMP_PROC_BIND=close
```

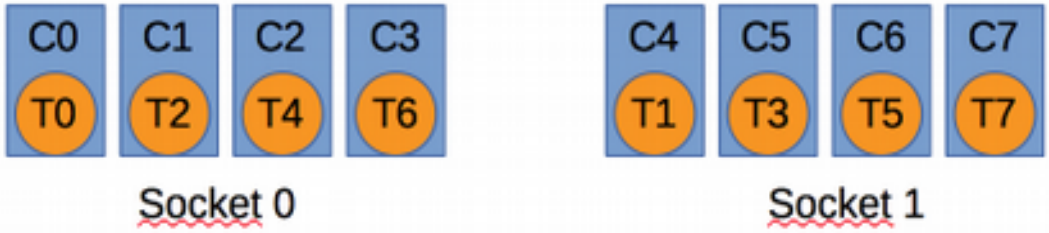
- Then
  - thread 0 goes to core 0, which is on socket 0,
  - thread 1 goes to core 1, which is on socket 0,
  - thread 2 goes to core 2, which is on socket 0,
  - and so on, until thread 3 goes to core 3 on socket 0, and
  - Thread 4 goes to core 4, which is on socket 1
  - et cetera.



Some examples

```
export OMP_PLACES=cores
export OMP_PROC_BIND=spread
```

- Then
  - thread 0 goes to socket 0,
  - thread 1 goes to socket 1,
  - thread 2 goes to socket 0 again,
  -



MHPC
Master in High Performance Computing

What is the difference between these two ?

```
OMP_PLACES=cores
OMP_PROC_BIND=spread
```

```
OMP_PLACES=sockets
```

The difference is that the latter choice does not bind a thread to a specific core, so the operating system can move threads about, and it can put more than one thread on the same core, even if there is another core still unused.

SISSA ICTP

MHPC
Master in High Performance Computing

Thread placement - OpenMP Affinity (Intel compiler)

- 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

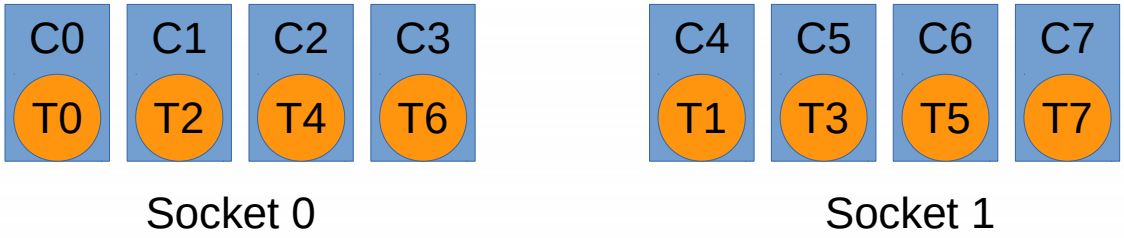
permute and offset
Both are integers
0

Defaults are **red**

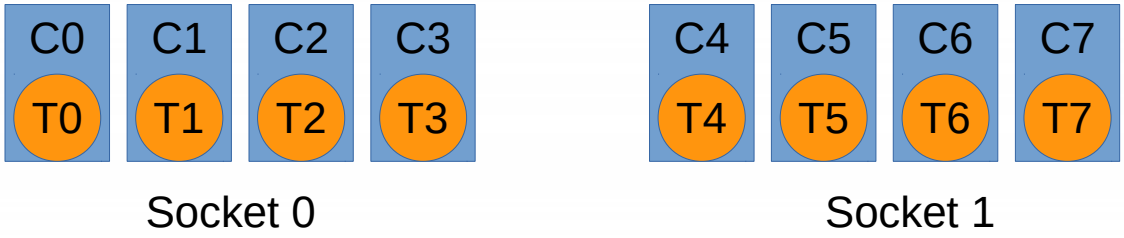
SISSA ICTP

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

- KMP_AFFINITY=scatter

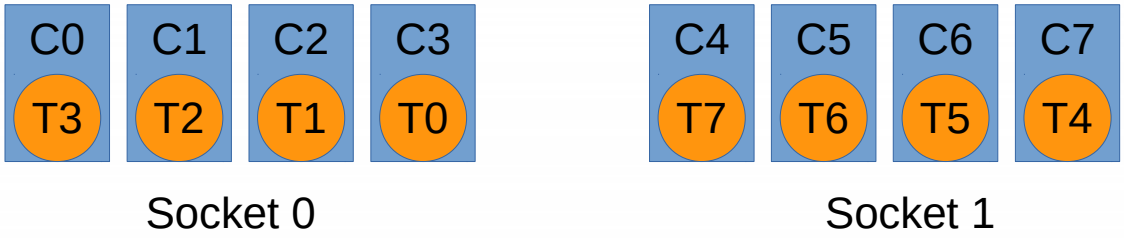


- KMP_AFFINITY=compact

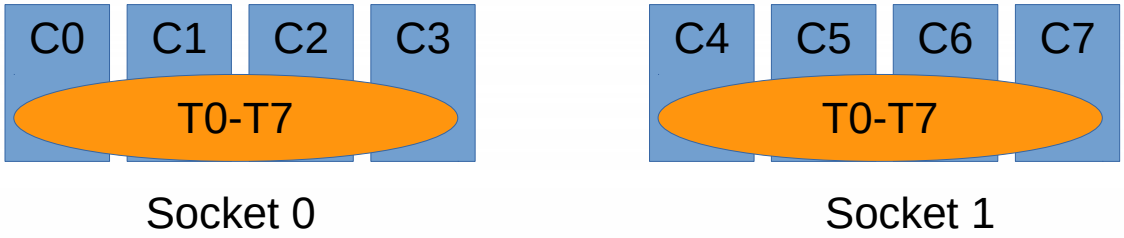


Thread placement - OpenMP Affinity (Intel)

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



- KMP_AFFINITY=none



MHPC
Master in High Performance Computing

hwloc

Portable Hardware Locality

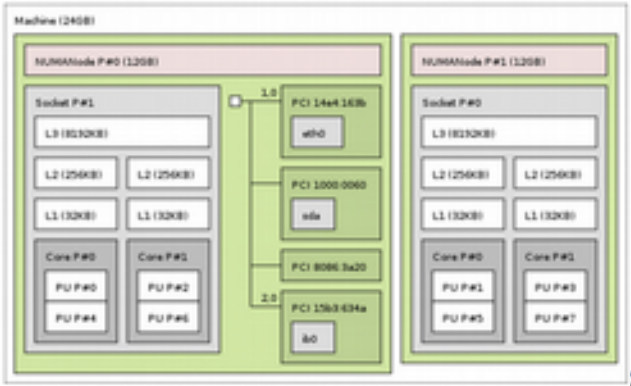
Portable topology information

Portable binding toolset

Portable Hardware Locality (hwloc)

Home | Support | FAQ | Search

The Portable Hardware Locality (hwloc) software package provides a portable abstraction (across OS, versions, architectures, ...) of the hierarchical topology of modern architectures, including NUMA memory nodes, sockets, shared caches, cores and simultaneous multithreading. It also gathers various system attributes such as cache and memory information as well as the locality of I/O devices such as network interfaces, InfiniBand HCAs or GPUs. It primarily aims at helping applications with gathering information about modern computing hardware so as to exploit it accordingly and efficiently.



The democratization of multicore processors and NUMA architectures leads to the spreading of complex hardware topologies into the whole server world. Nodaways every single cluster node may contain tens of cores, hierarchical caches, and multiple memory nodes, making its topology far from flat. Such complex and hierarchical topologies have strong impact of the application performance. The developer must take hardware affinities into account when trying to exploit the actual hardware performance. For instance, two tasks that tightly cooperate should probably rather be placed onto cores sharing a cache. However, two independent memory-intensive tasks should better be spread out onto different sockets so as to maximize their memory throughput. As described in this paper, OpenMP threads have to be placed according to their affinities and to the hardware characteristics. MPI implementations apply similar techniques while also adapting their communication strategies to the network locality as described in this paper or this one.

hwloc v1.11.0 published

New feature release

> Read more

SISSA

ICTP

MHPC
Master in High Performance Computing

hwloc

- Two parts
  - - Set of command line tools (lstopo, hwloc-bind, calc, etc.)
  - - C API + library, Perl and Python bindings
- Portable: Linux, Solaris, AIX, HP-UX, FreeBSD, Darwin, Windows
- BSD-3 license
- Used by a lot of projects: most MPI, runtimes, batch scheds, ...

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

SISSA ICTP

MHPC
Master in High Performance Computing

Binding processes and memory

`hwloc-bind – bind process`

Bind a new process to a given set of CPUs:

```
$ hwloc-bind socket:1 -- mycommand
```

Bind an existing process:

```
$ hwloc-bind --pid 1234 socket:1
```

Bind memory:

```
$ hwloc-bind --membind node:1 --cpubind node:1.socket:0
./a.out
```

Distribute memory:

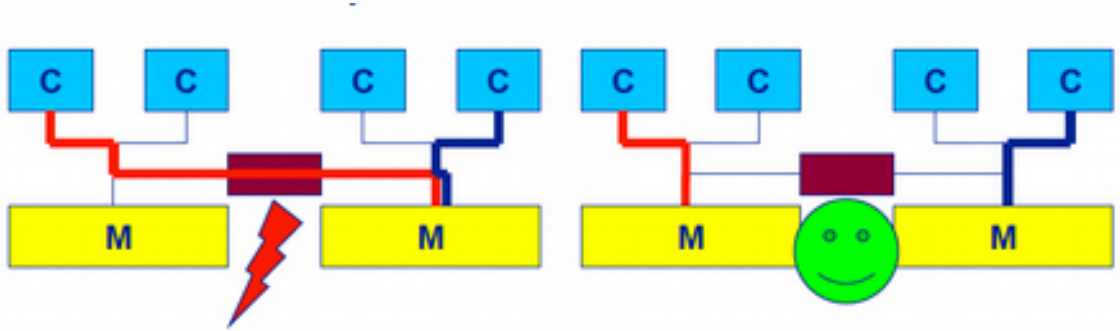
```
$ hwloc-bind --membind --mempolicy interleave all --
mycommand
```

SISSA ICTP

ccNUMA performance problem

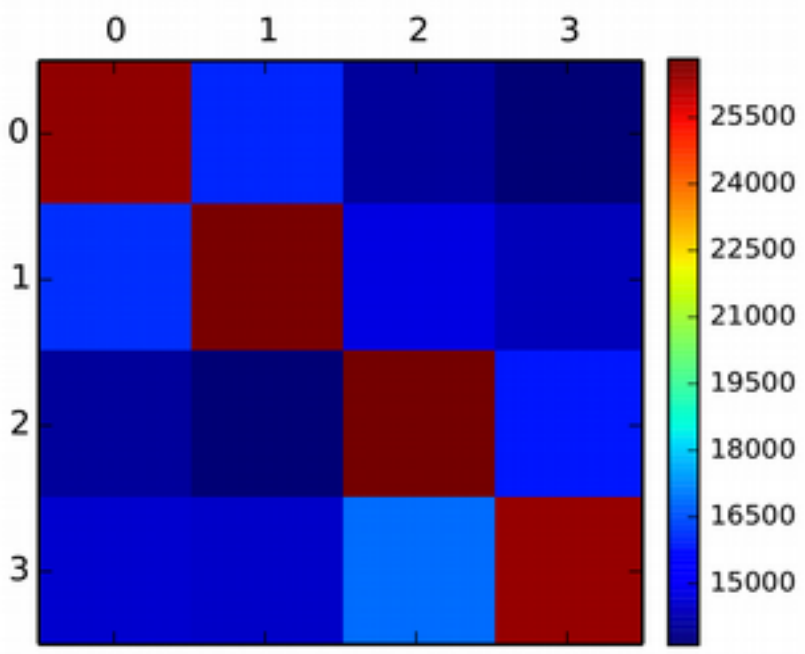
- CcNUma:
  - Whole memory is transparently accessible
  - But physically distributed with different latency and bandwidth

    => potential contention
- How to we make sure that memory access is always as “local” and “distribute” as possible ?



An example: Sandybridge node

- CcNuma map: bandwidth for remote access
  - Run 10 threads per ccNUMA domain
  - Place memory in different domain



MHPC
Master in High Performance Computing

Numactl as simple ccNuma locality tool:

- Numactl can influence the way a code maps the memory pages

```
- Numactl   –membind=<nodes> a.out     # map page on
                                       # <node>
-           --preferred=<nodes> a.out  # map pages on
                                       # <node> and the
                                       # when it is full
                                       #  on others
          -- interleaved= <nodes> a.out
```

- Example:

```
for m in `seq 0 3`; do
  for c in `seq 0 3`; do
    env OMP_NUM_THREADS=8 \
        numactl --membind=$m --cpunodebind=$c ./stream
  enddo
enddo
```

ccNUMA map scan

CcNUma default memory locality

- Golden Rule: " a memory page gets mapped into the local memory of the processor that first touches it
- "Touch" means "write" not" allocate"
- It is enough to write a single element to map an entire page
- Example:

```
double *huge = (double*)malloc(N*sizeof(double));

for(i=0; i<N; i++) // or i+=PAGE_SIZE
   huge[i] = 0.0;
```

Mapping takes place here

Coding for data locality:

- Simplest case: explicit initialization

```
integer,parameter :: N=10000000
double precision A(N), B(N)



A=0.d0



!$OMP parallel do
do i = 1, N
  B(i) = function ( A(i) )
end do
!$OMP end parallel do
```

```
integer,parameter :: N=10000000
double precision A(N),B(N)
!$OMP parallel
!$OMP do schedule(static)
do i = 1, N
  A(i)=0.d0
end do
!$OMP end do
...
!$OMP do schedule(static)
do i = 1, N
  B(i) = function ( A(i) )
end do
!$OMP end do
!$OMP end parallel
```

MHPC
Master in High Performance Computing

Summary on thread placement

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

SISSA ICTP

MHPC
Master in High Performance Computing

And now...

- Stream : see stream
- MPI benchmarks: mpi
- Nodeperf: nodeperf

SISSA ICTP