MHPC
Master in High Performance Computing

# P1.2 course:
# 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
- How to program and to exploit computing power
  - Optimized multithreaded library
- Evaluate performance of multicore node (tutorial)
  - Stream to measure memory
  - MPI benchmark to measure latency among different cores
  - Nodeperf code
  - HPL using multithreaded library

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
- Understand basic principle of multithreading and multicore  by means of Threaded libraries

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

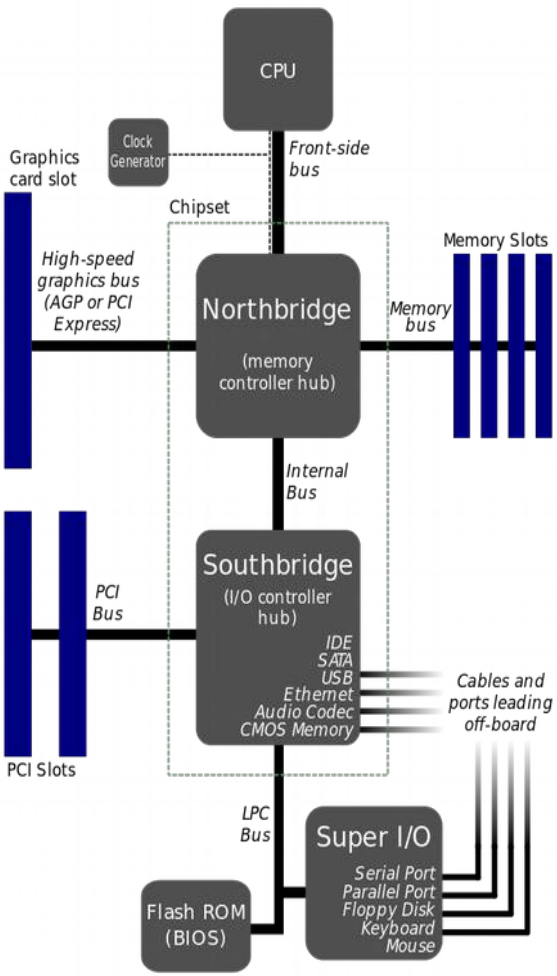
MHPC
Master in High Performance Computing

## 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

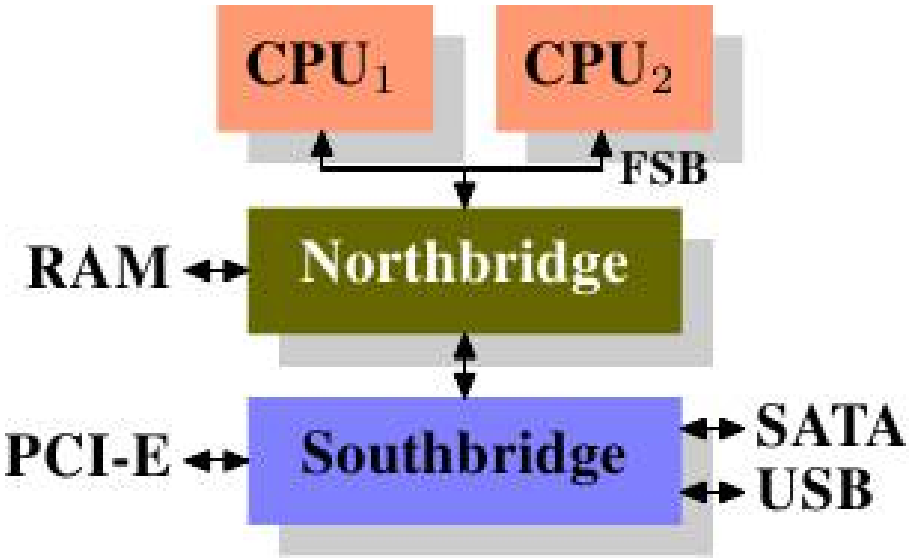
## standard modern architecture

- 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.



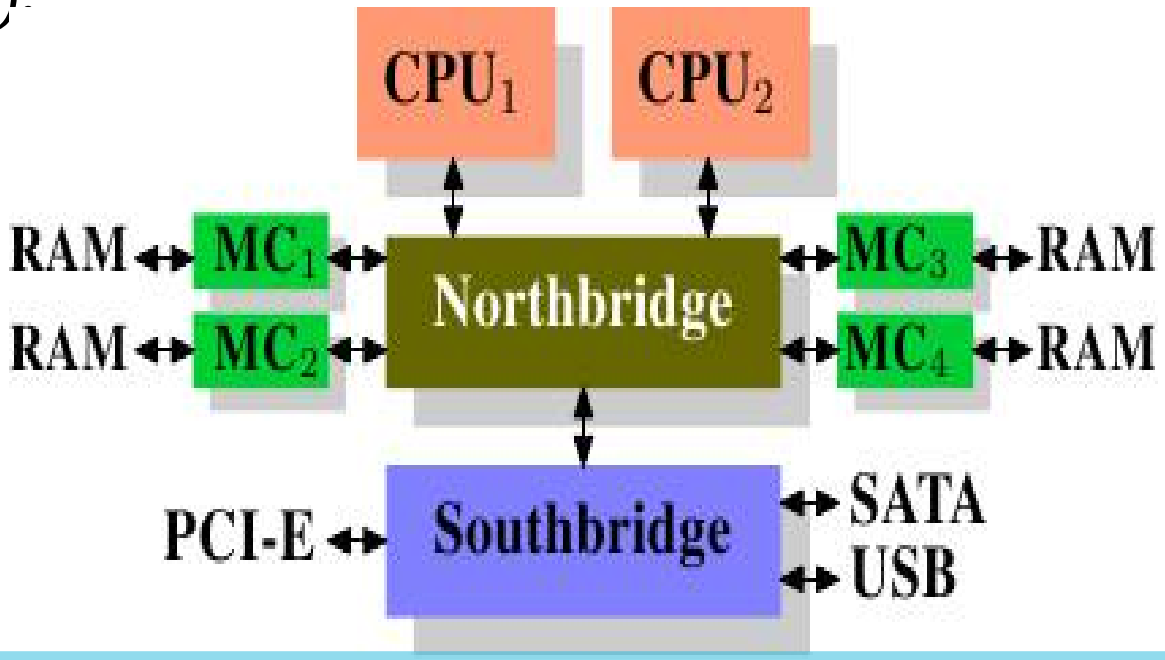
## standard multisocket architecture

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



**more expensive and modern architecture**

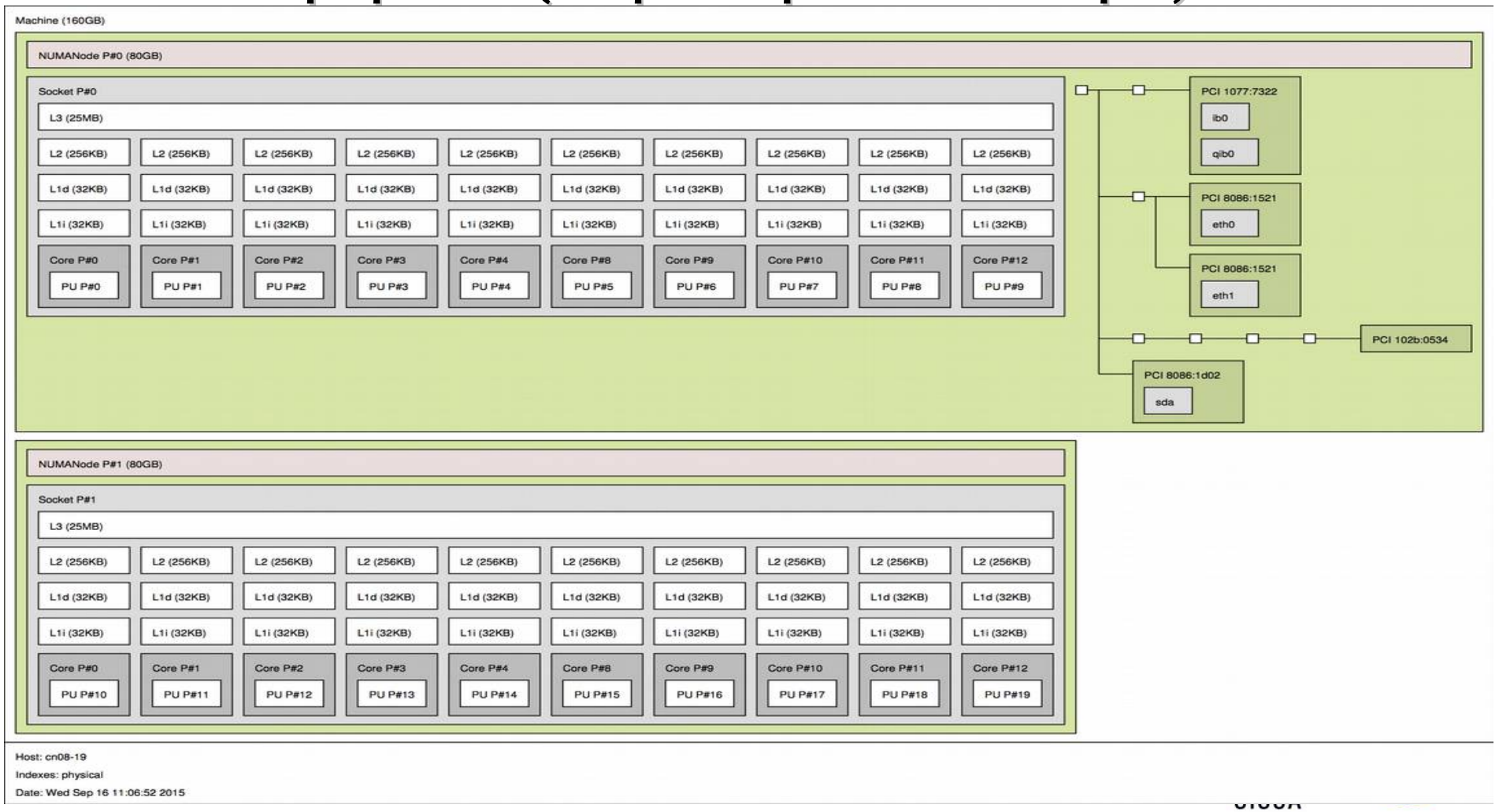
- Northbridge can be connected to a number of external memory controllers (in the following example, four of them)




INCREASE IN BANDWIDTH TOWARD MEMORY

MHPC
Master in High Performance Computing

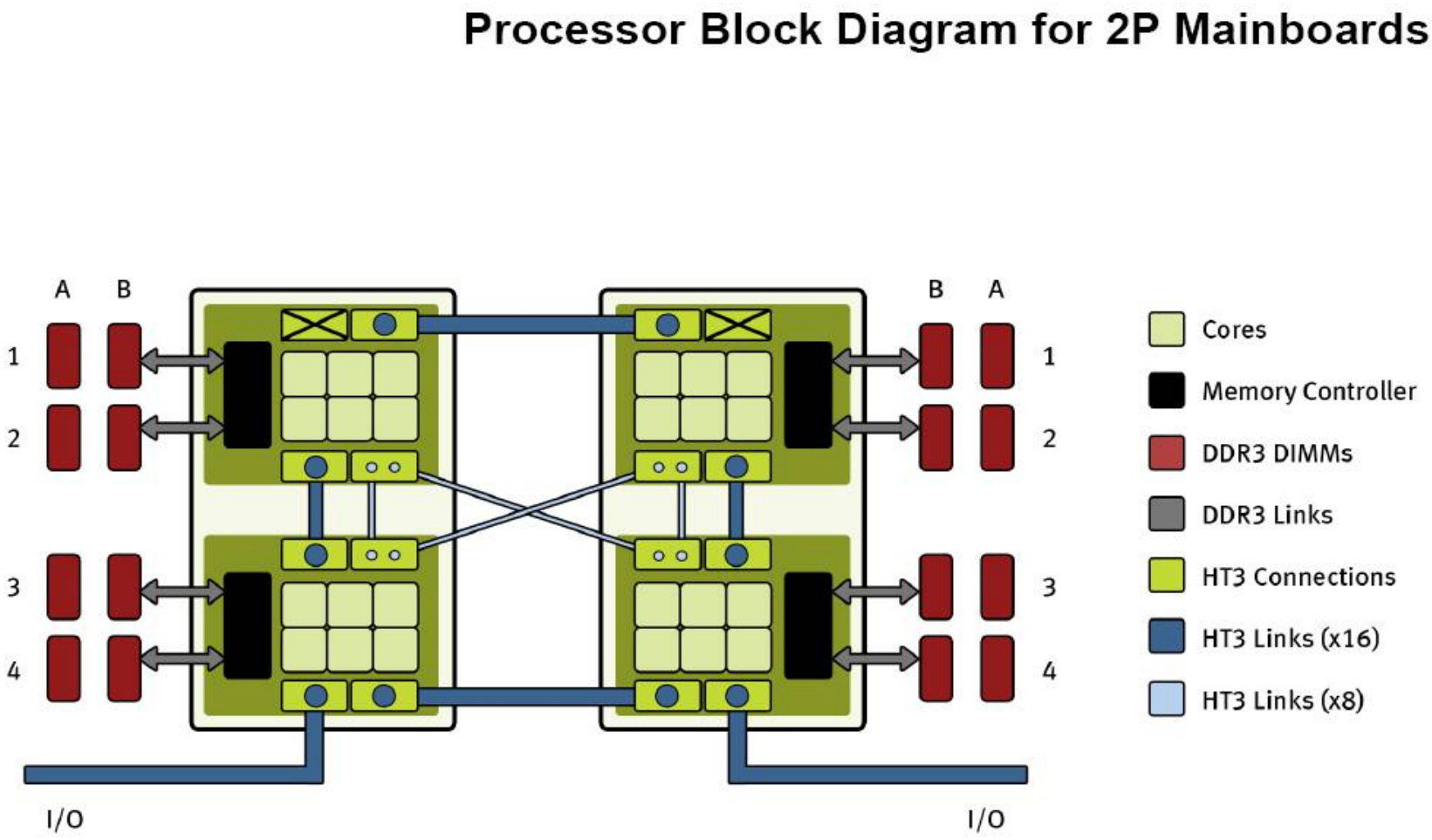
# A modern cpu picture (lstopo –of pdf >out-node.pdf)



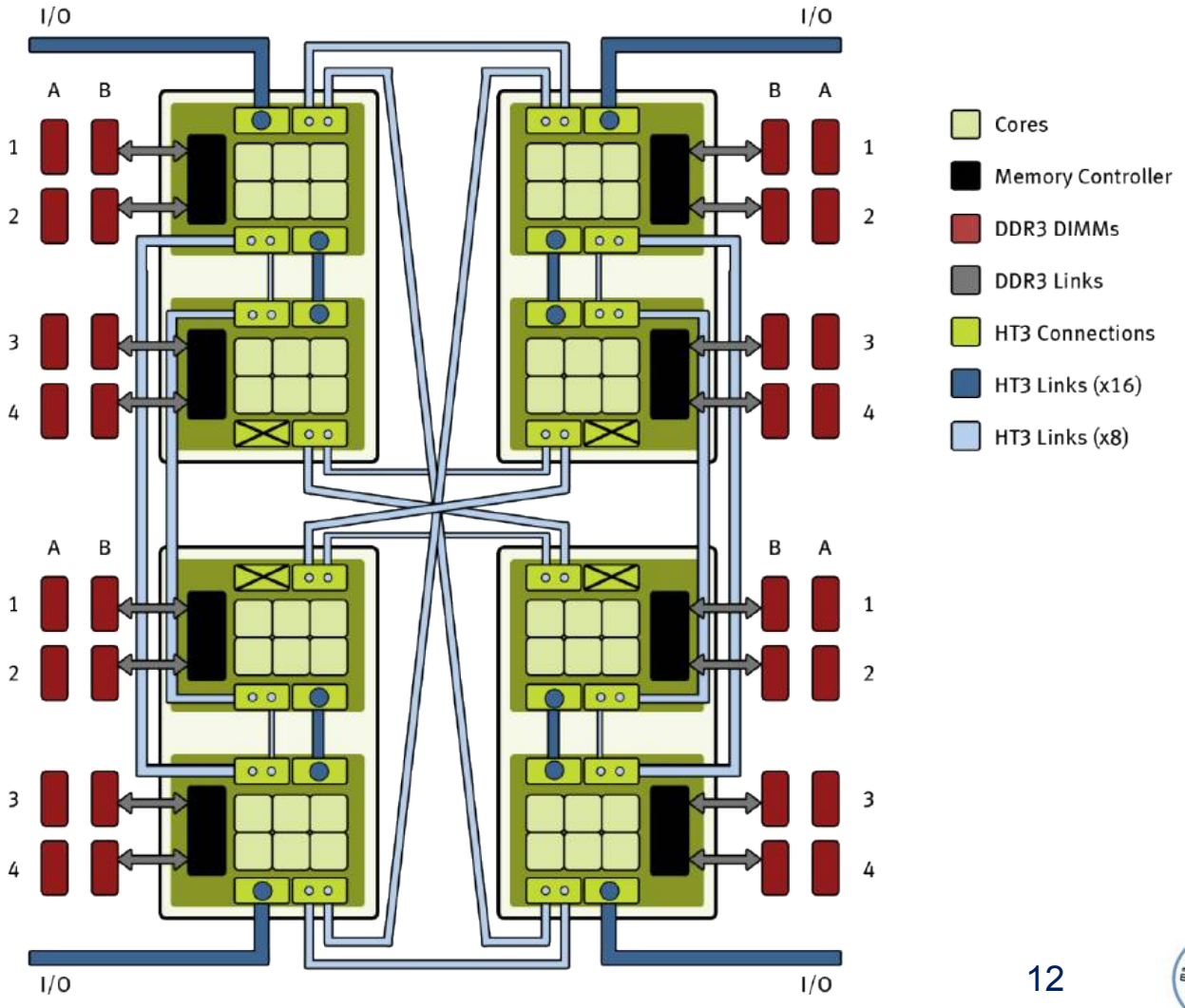
## From SMP to NUMA

- 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)

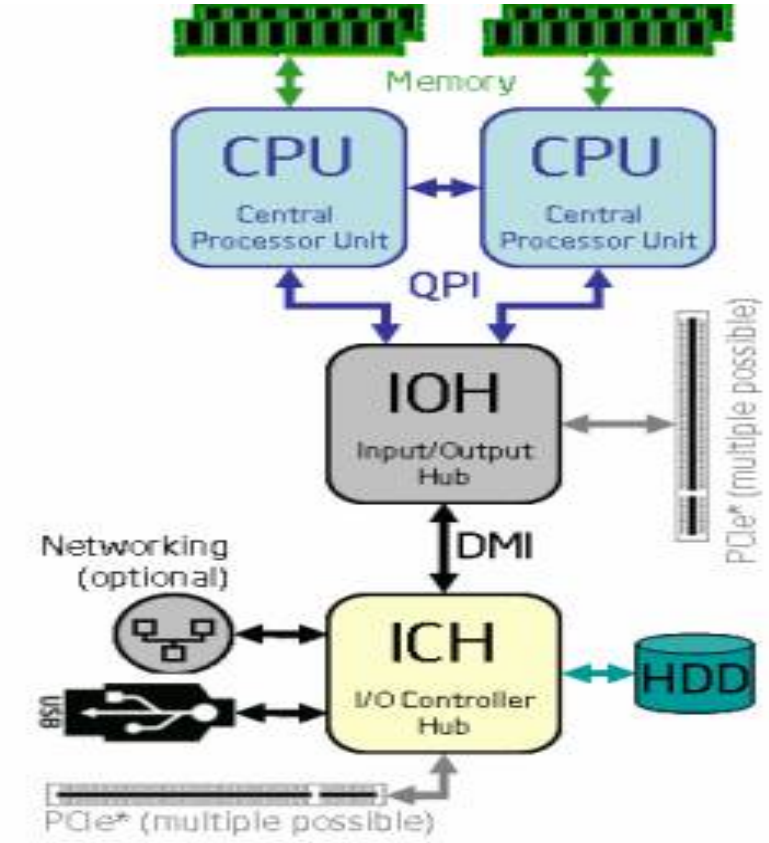
# Opteron 6xxx AMD CPU



# Processor Block Diagram for 4P Mainboards



# Xeon Family: Nehalem introduces NUMA



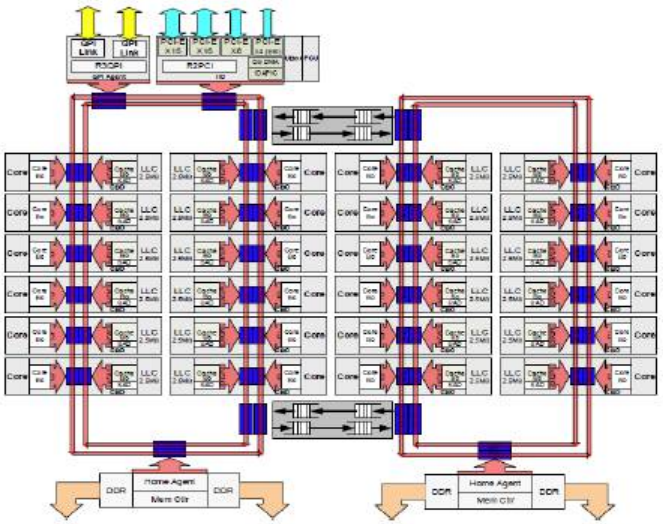

- First NUMA architecture by INTEL
- QPI among CPUs to play the role of hyper-transport in AMD
- Released April 2009

MHPC
Master in High Performance Computing

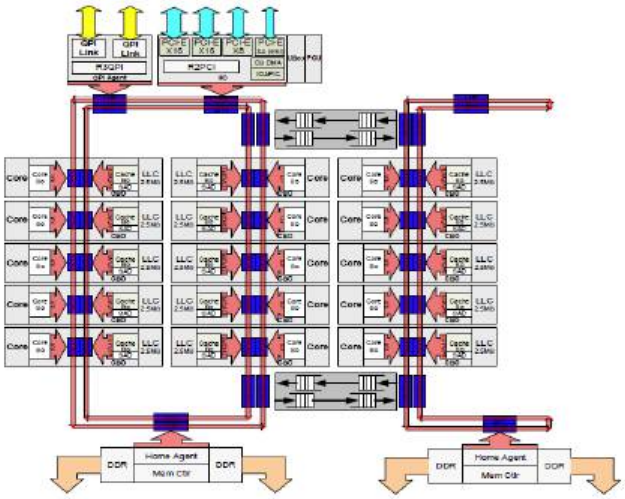
# Brodwell layout

## Broadwell EP die configurations

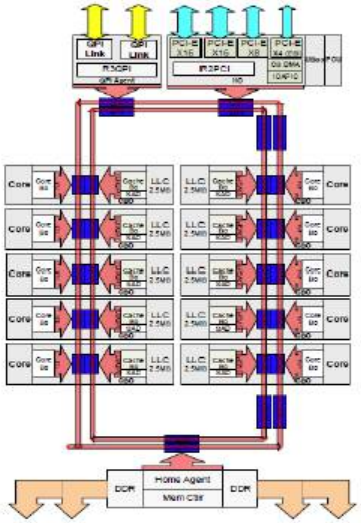
Up to 22 Core (HCC)



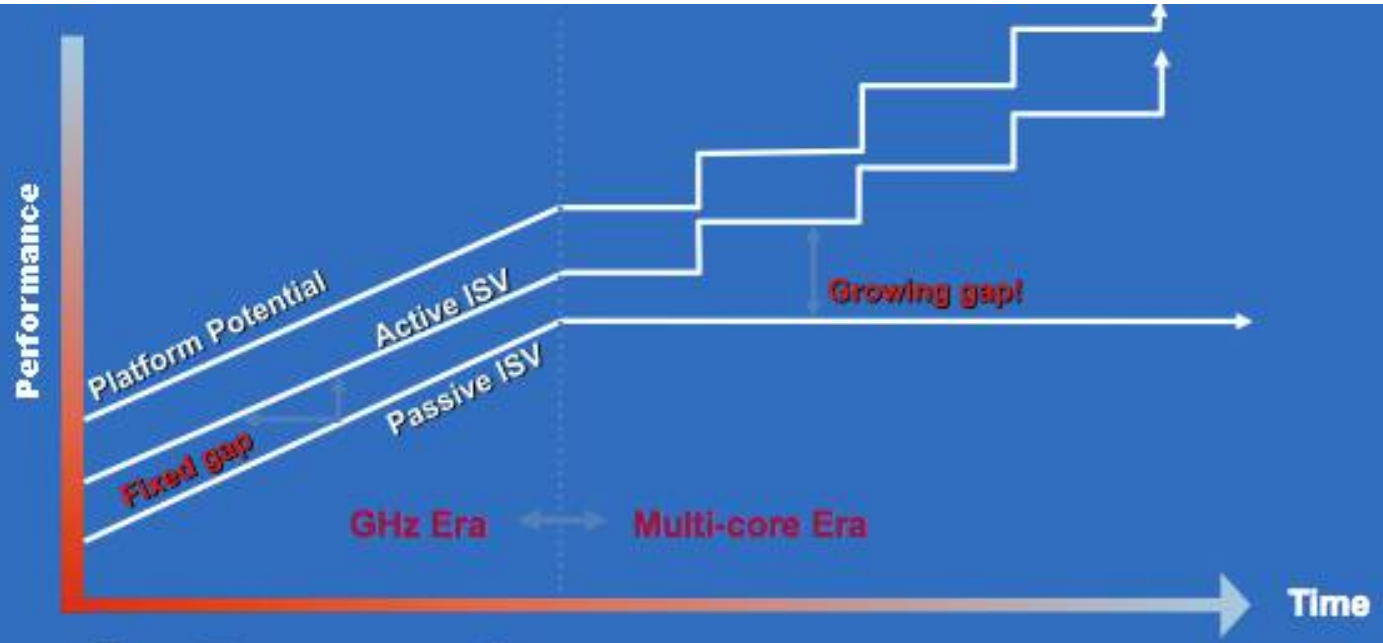
Up to 14 Core (MCC)



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 |




## "Parallelism for Everyone"

Parallelism changes the game

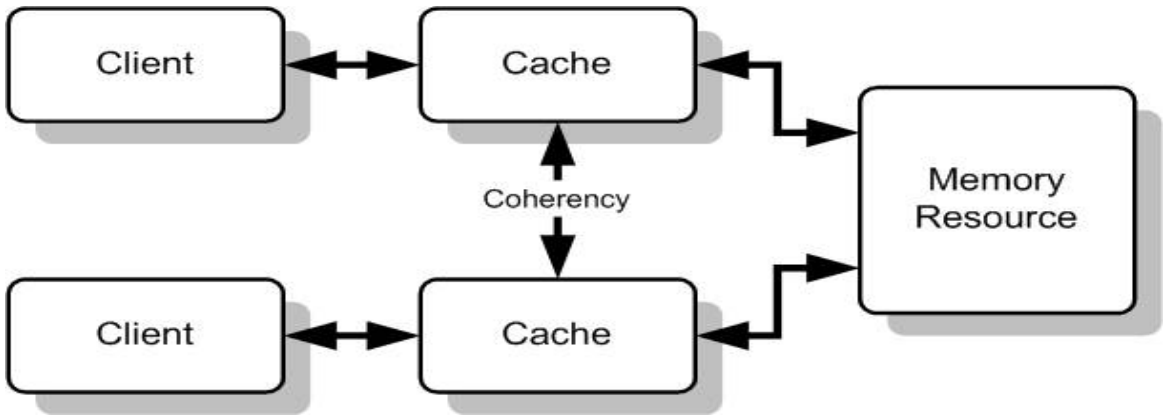
- A large percentage of people who provide applications are going to have to care about parallelism in order to match the capabilities of their competitors.

## Challenges for multicore

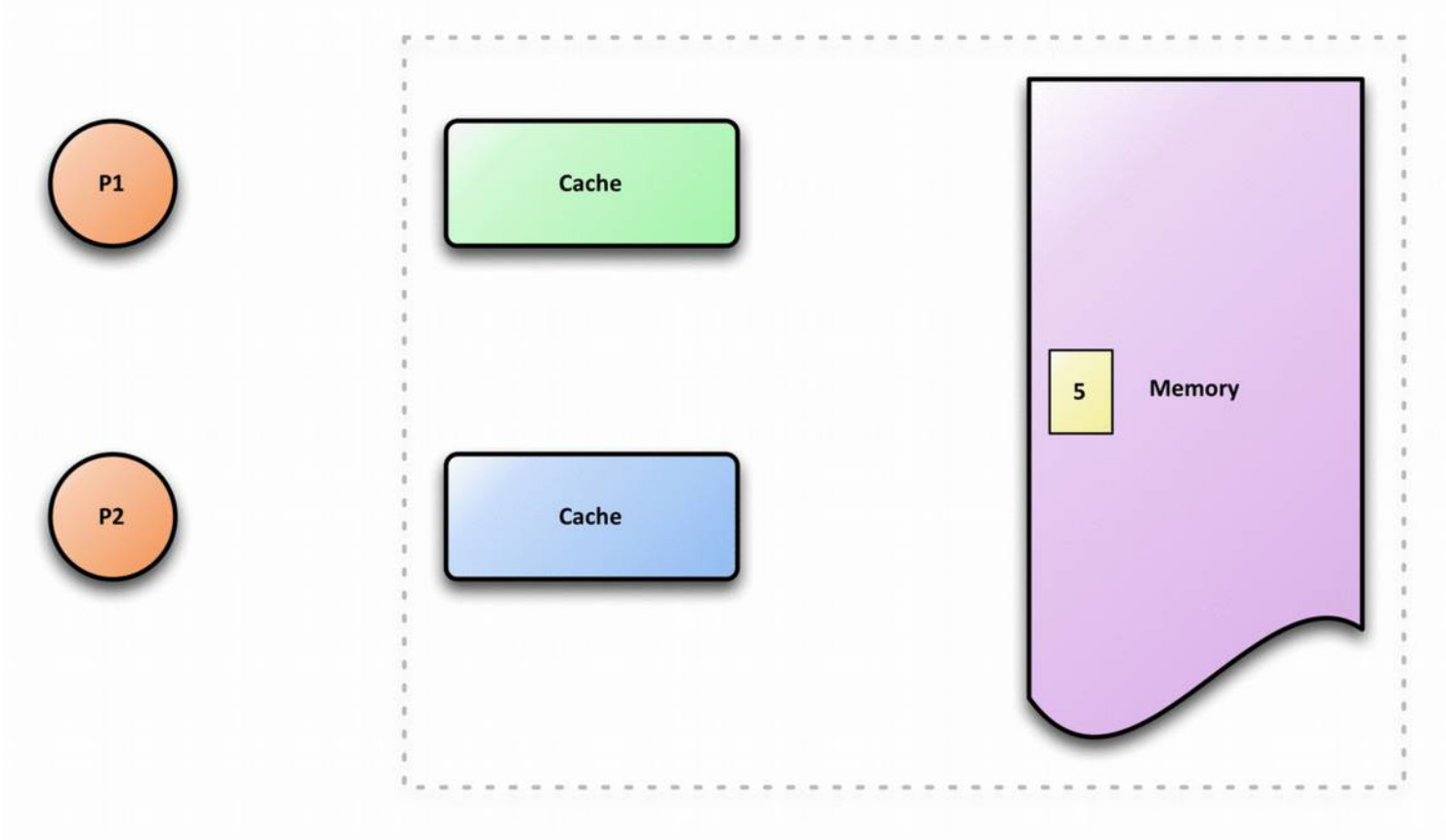
- Relies on effective exploitation of multiple-thread parallelism
  - Need for parallel computing model and parallel programming model
- Aggravates memory wall problem
  - Memory bandwidth
    - Way to get data out of memory banks
    - Way to get data into multi-core processor array
    - Memory latency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

## Cache coherency

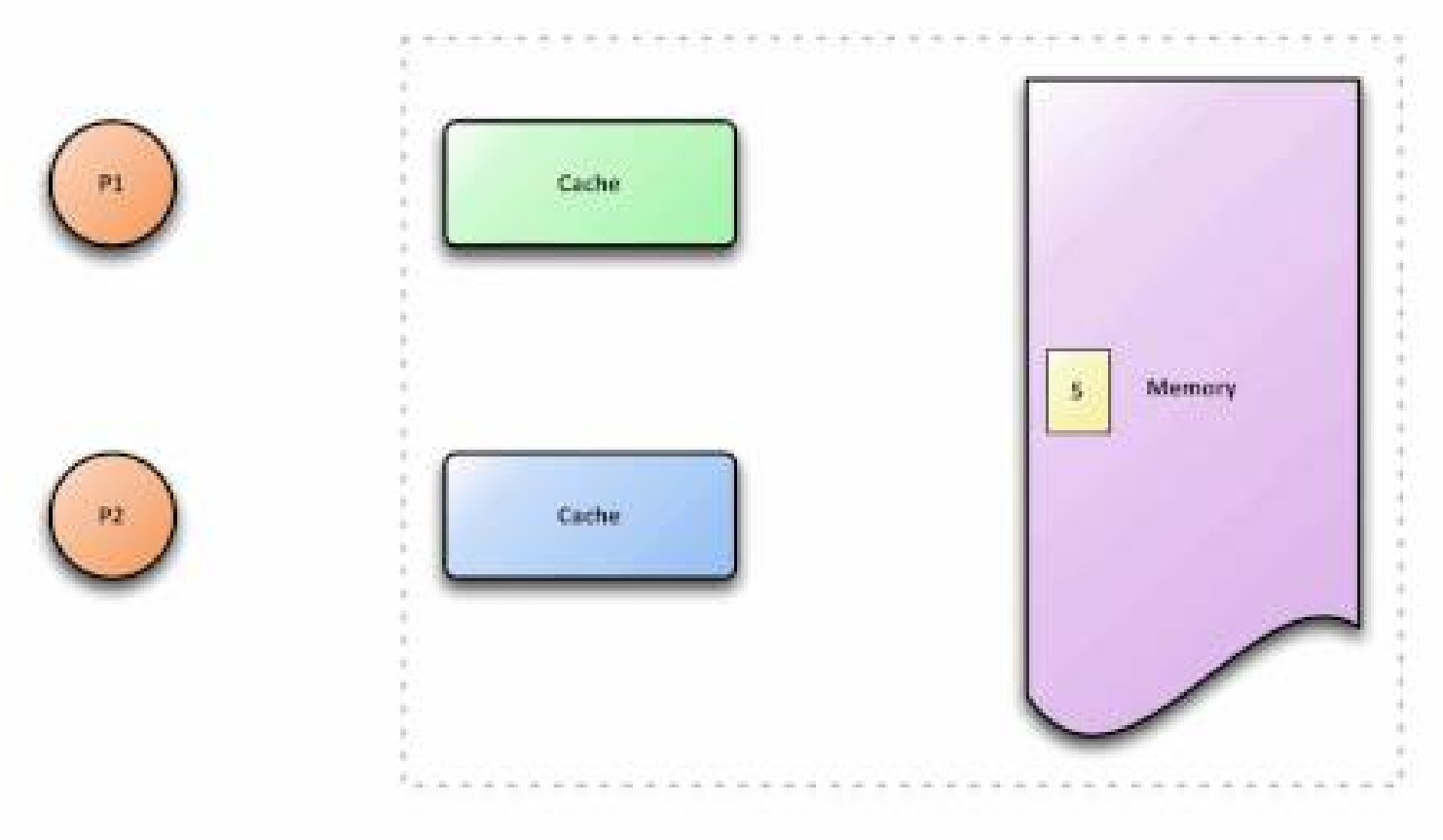
- From wikipedia:
  - the uniformity of shared resource data that ends up stored in multiple local caches. When clients in a system maintain caches of a common memory resource, problems may arise with incoherent data, which is particularly the case with CPUs in a multiprocessing system.



# 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];
```

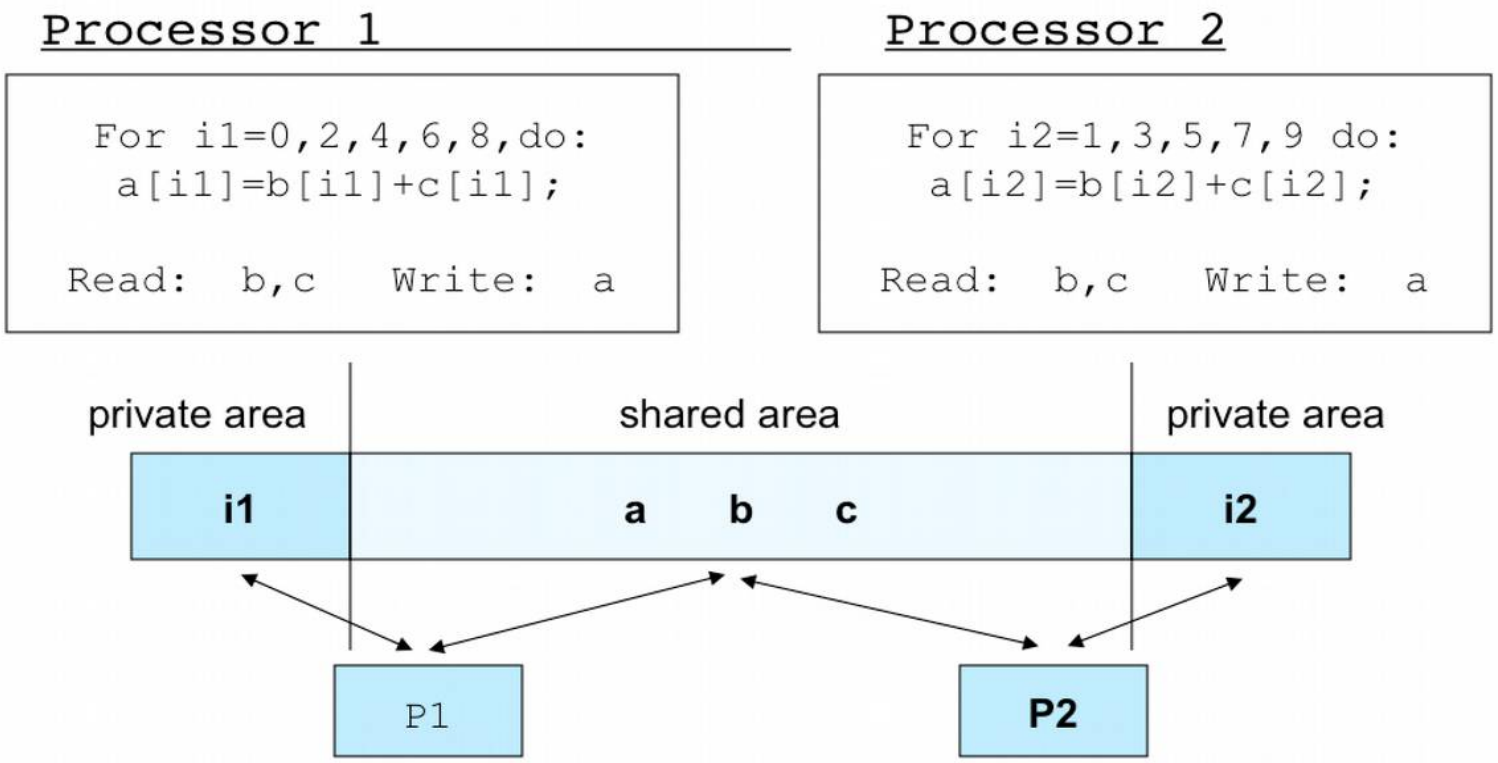
- Let's assume we run this on 2 processors:

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

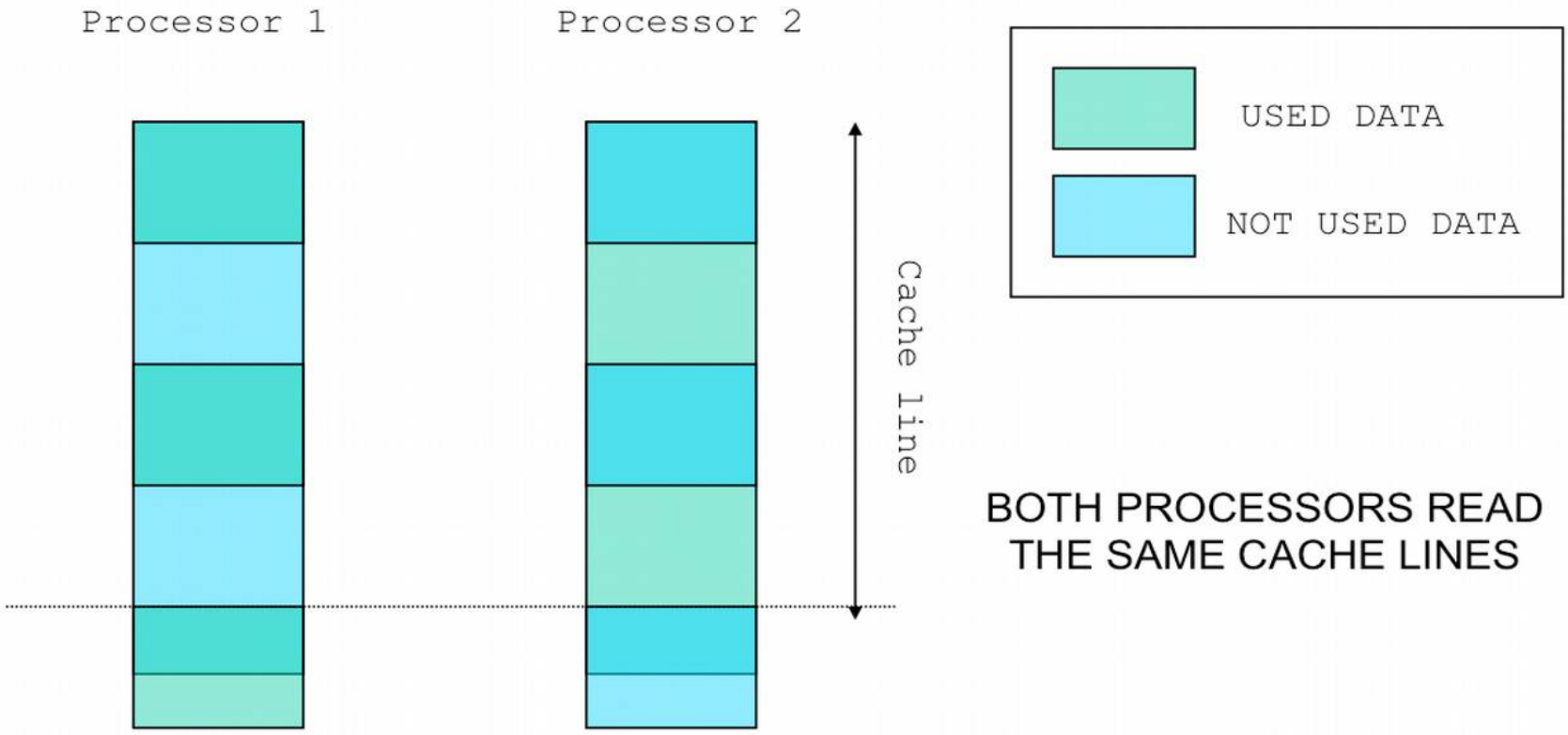
processor 2  for  I=1,3,5,7,9

SISSA ICTP

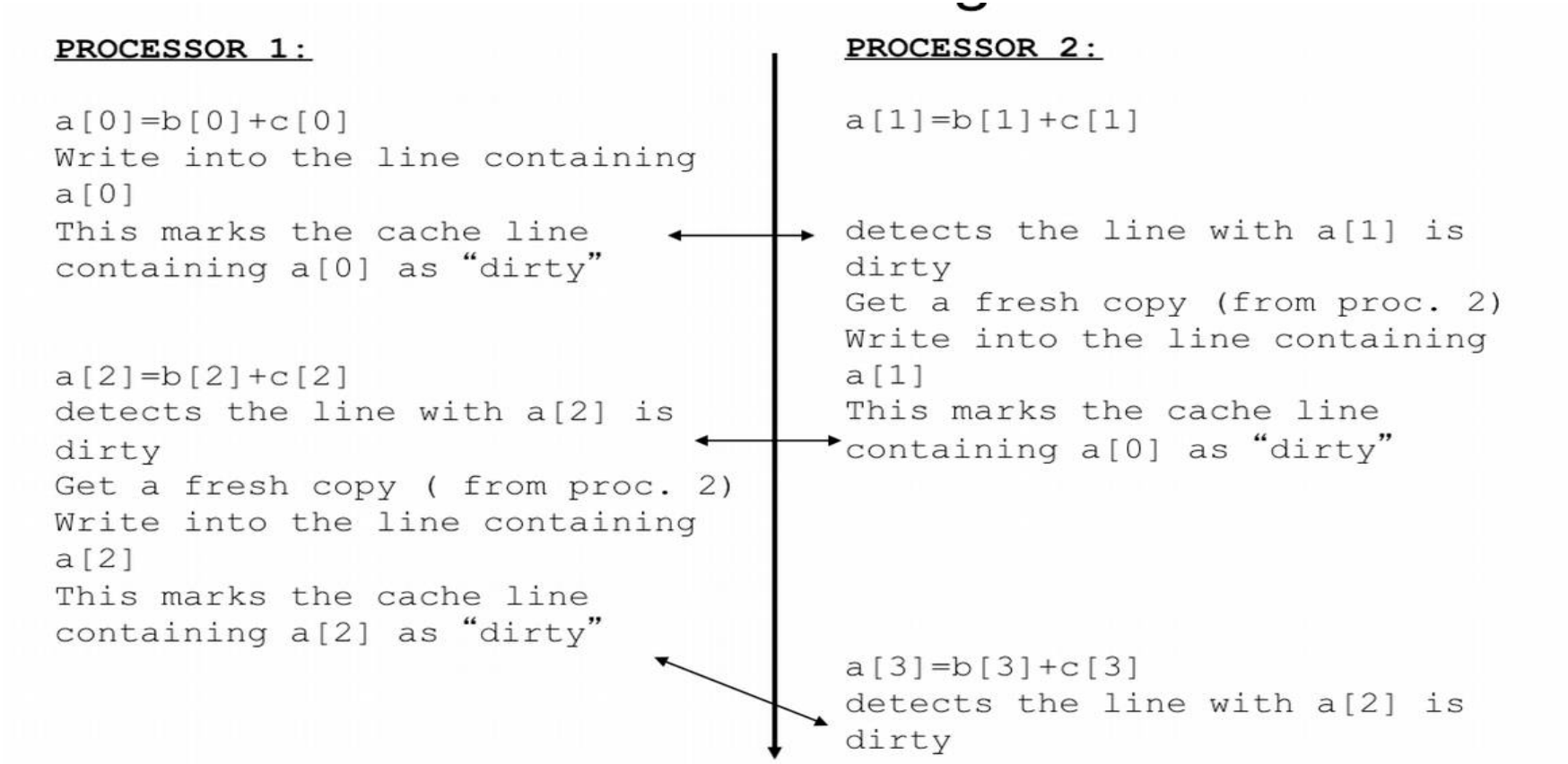
# What is happening ? (1)



# What is happening ? (2)



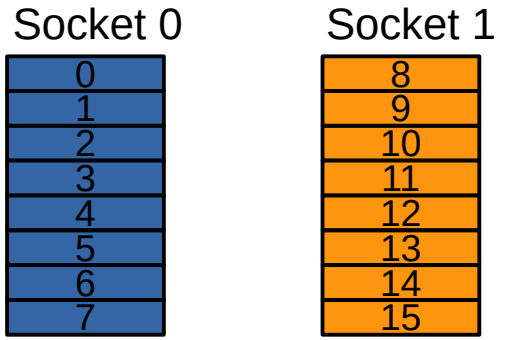
# What is happening ? (3)



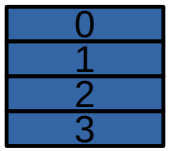
# Share memory - NUMA

- Dual socket nodes can operate in two modes

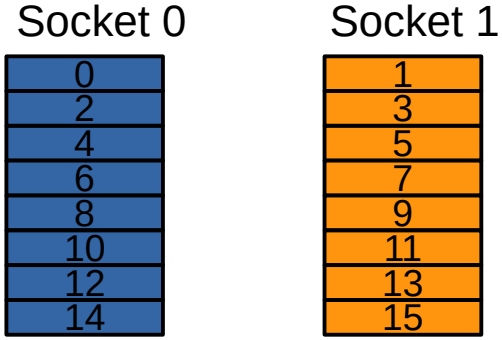
**NUMA**
(numa=on)




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




**Interlaced**
(numa=off)



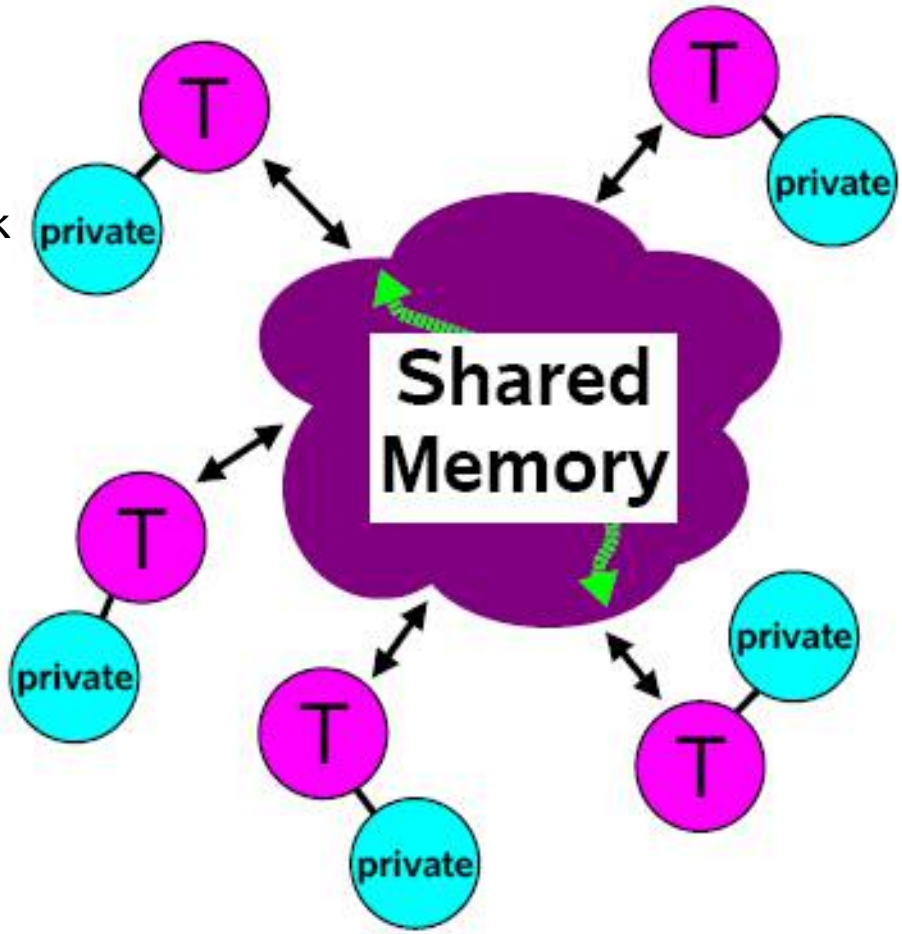

Memory controller maps each other cache-line from another socket to contiguous memory. Allocated memory might look like this:


NUMA mode generally gives the best results for HPC

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

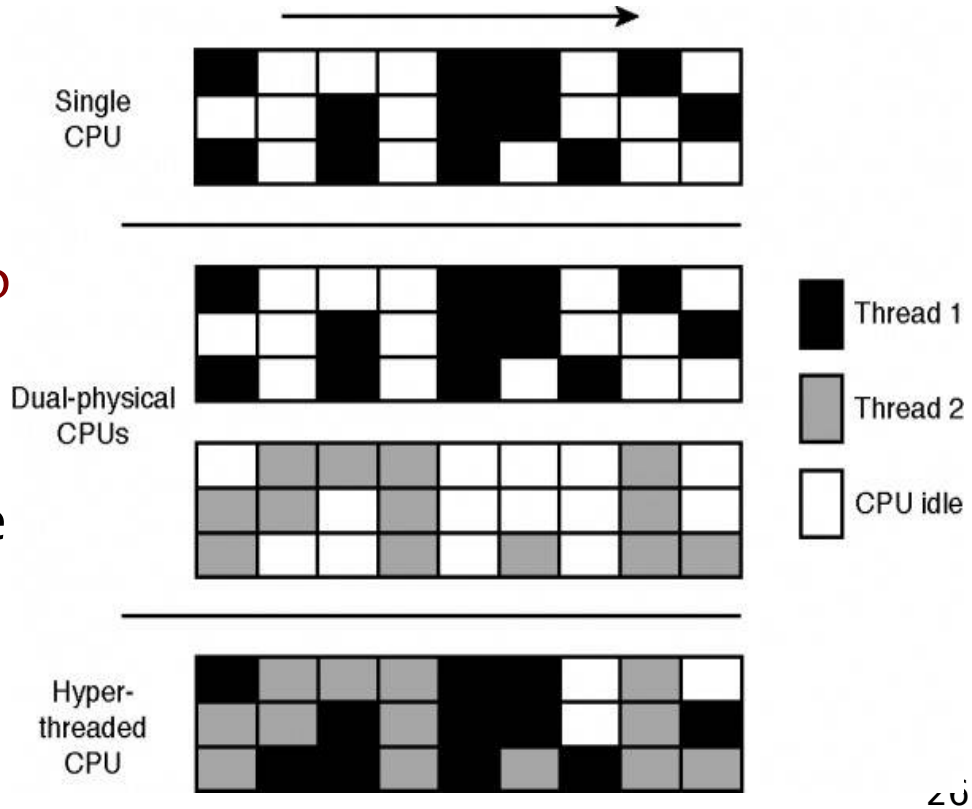
# the shared programming model

- 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



## 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..
  - Why ???



## hello world in openMP

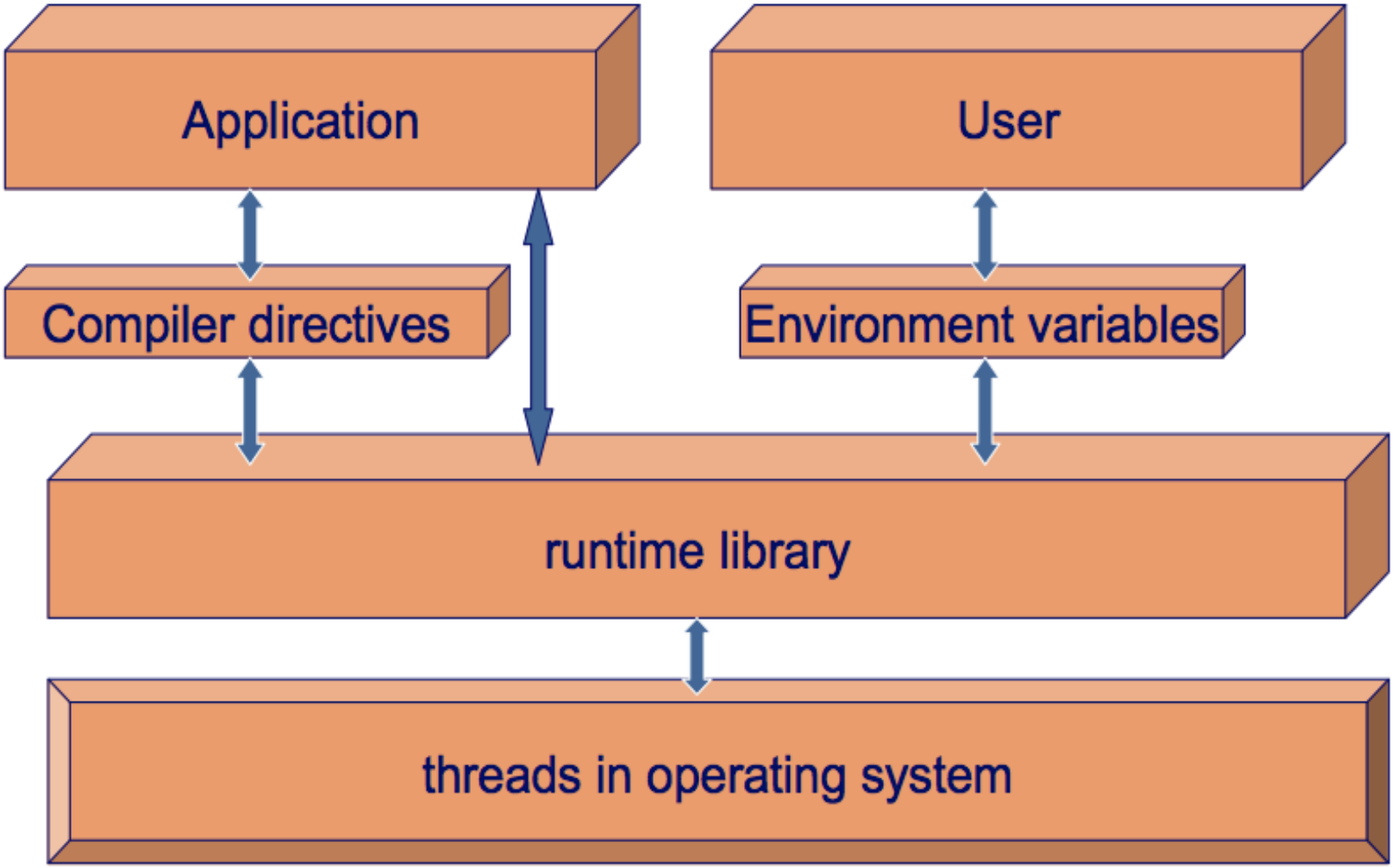
```
#include <omp.h>
int main()  {
  int iam =0, np = 1;
#pragma omp parallel private(iam, np)
  {
#if defined (_OPENMP)
   np = omp_get_num_threads();
   iam = omp_get_thread_num();
#endif
  printf("Hello from thread %d out of %d \n", iam, np);
 }
}
```

## 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

## 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)
```

## 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]$ 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
```

MHPC
Master in High Performance Computing

## Optimization Techniques

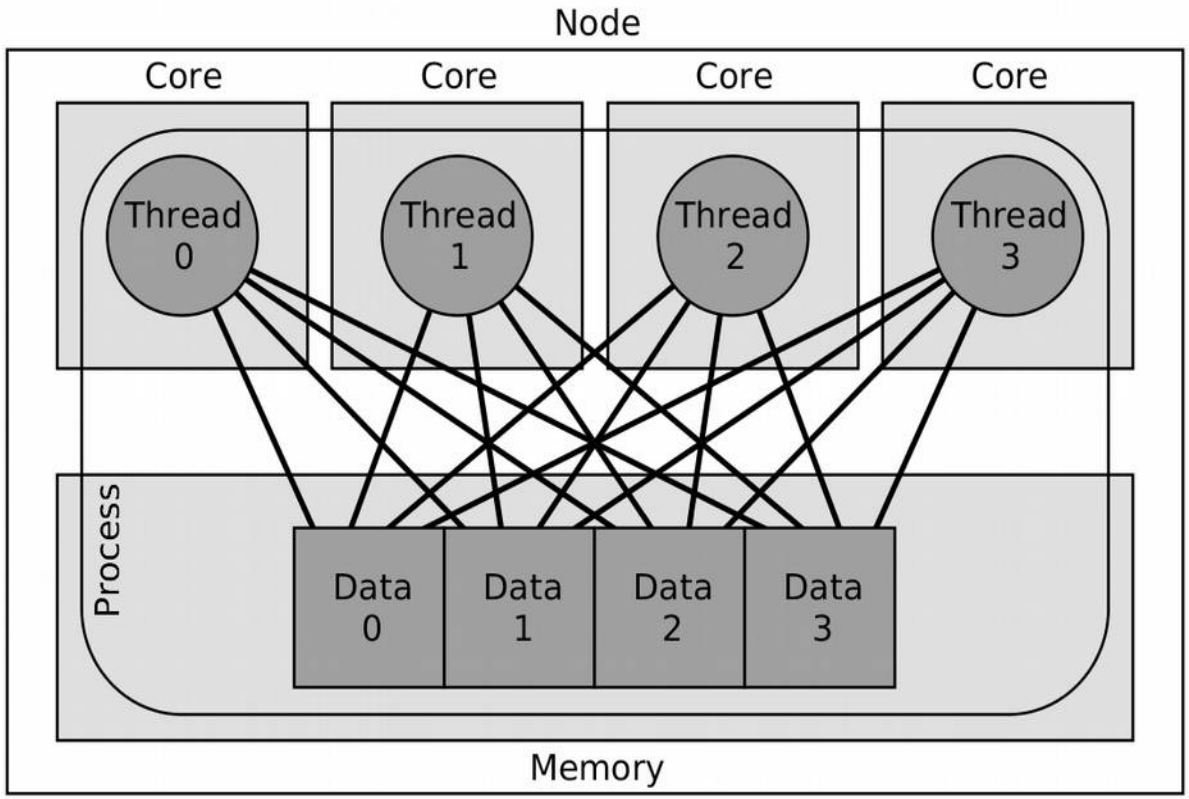
- There are basically three different categories:

1
  - Improve memory performance (The most important)
  - Improve CPU performance

2
  - Use already highly optimized libraries/subroutines

SISSA
ICTP

# Threads 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

MHPC
Master in High Performance Computing

## Thread Placement

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

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

SISSA ICTP

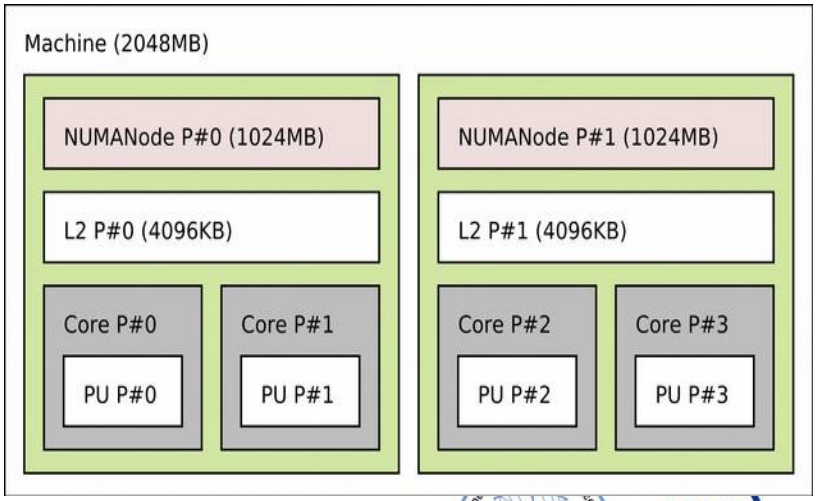
MHPC
Master in High Performance Computing

## 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 - 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

# 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
```

## 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.

## 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.

MHPC
Master in High Performance Computing

## Thread placement in OpenMP

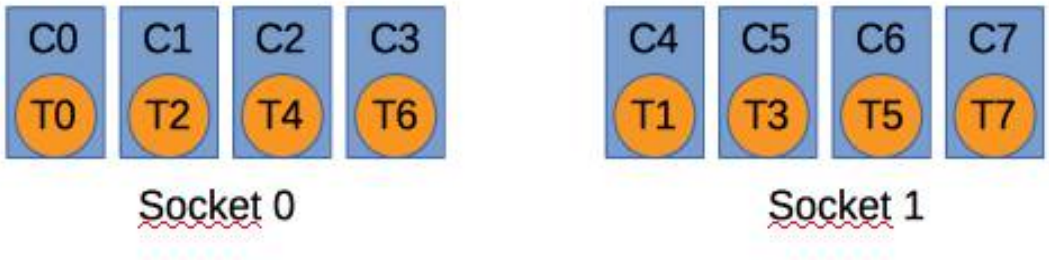
- Thread placement can be controlled with two environment variables:
- the environment variable OMP_PROC_BIN describes how threads are bound to OpenMP places
- the variable OMP_PLACES describes these places in terms of the available hardware.
- When you're experimenting with these variables it is a good 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
```

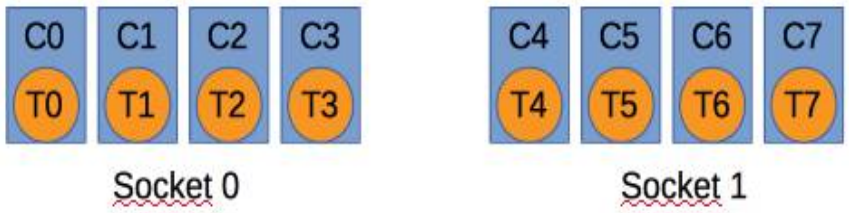
- Then
  - thread 0 goes to socket 0,
  - thread 1 goes to socket 1,
  - thread 2 goes to socket 0 again,
  - and so on.



## Some examples

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

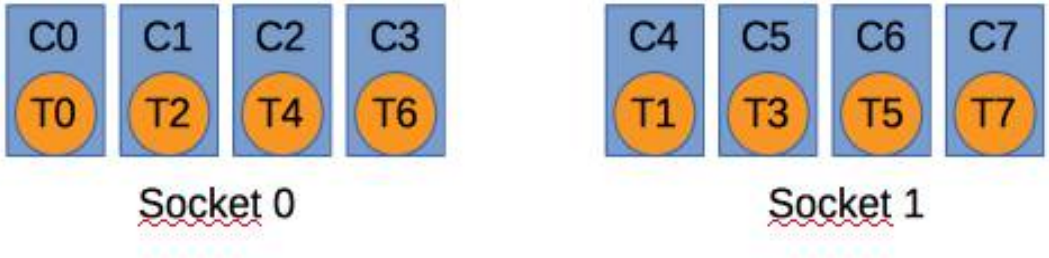
- Then
  - thread 0 goes to core 0, which is on socket 0,
  - thread 1 goes to core 1, which is on socket 0,
  - thread 2 goes to core 2, which is on socket 0,
  - and so on, until thread 3 goes to core 3 on socket 0, and
  - Thread 4 goes to core 4, which is on socket 1
  - et cetera.



## 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**

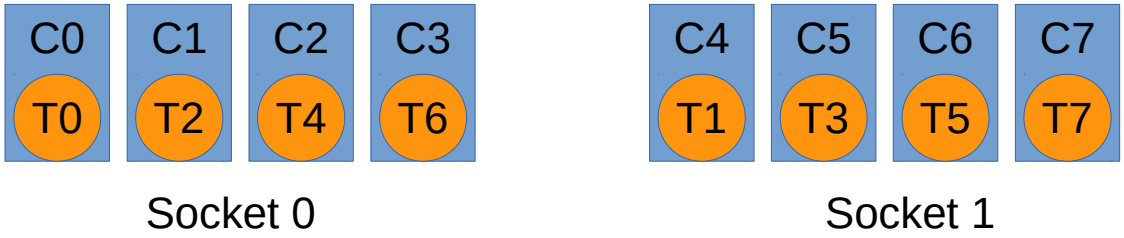
Defaults are **red**

SISSA ICTP

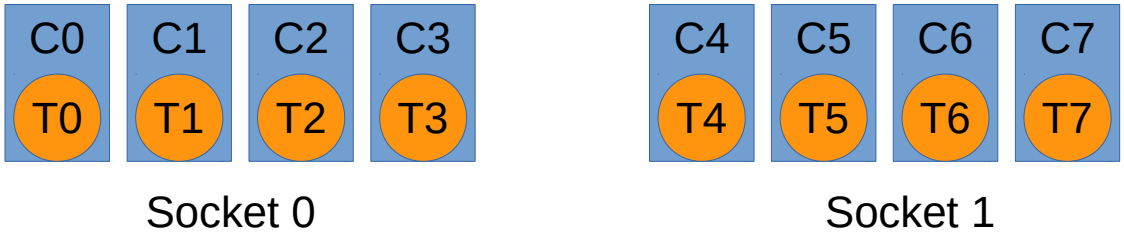
MHPC
Master in High Performance Computing

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

- KMP_AFFINITY=scatter



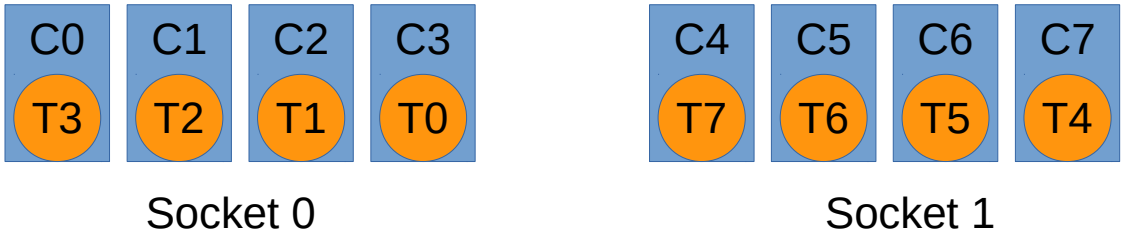

- KMP_AFFINITY=compact



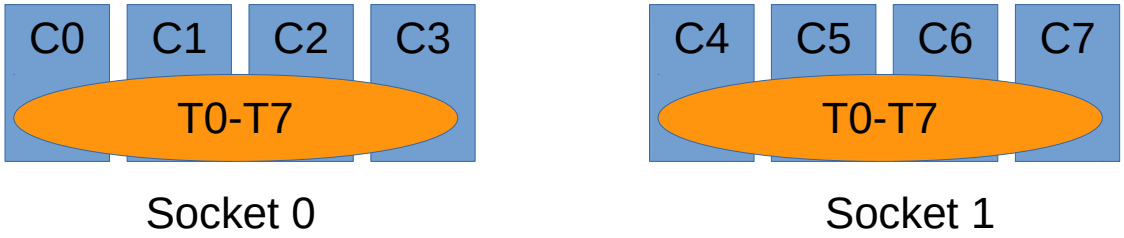

SISSA
ICTP

# 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

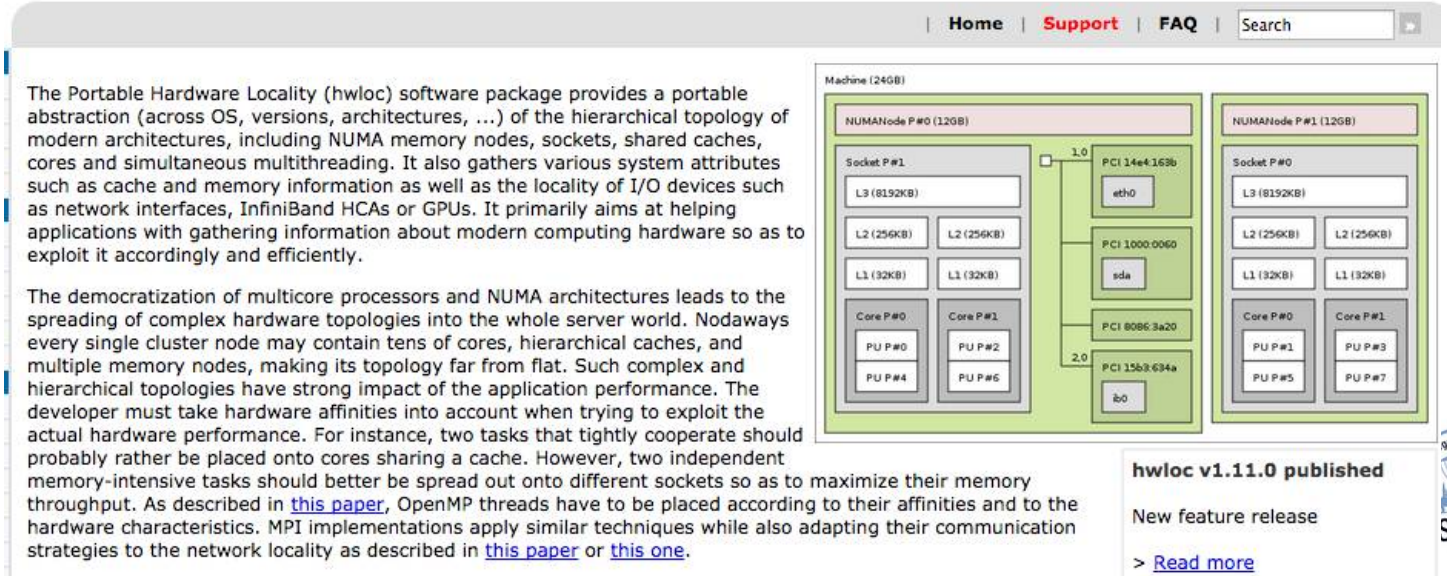
Portable Hardware Locality

Portable topology information

Portable binding toolset

---

## Portable Hardware Locality (hwloc)




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

## Let us use it..

lstopo – Displaying topology information

hwloc-distances – show object distances

Notably NUMA distances:

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

# 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
```

MHPC
Master in High Performance Computing

## Summary on thread placement

- In (ubiquitous) NUMA systems, proper thread and process placement is a must
- Numctl 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

## What are performance libraries ?

- Routines for common (math) functions such as vector and matrix operations, fast Fourier transform etc. written in a specific way to take advantage of capabilities of the CPU.
- Each CPU type normally has its own version of the library specifically written or compiled to maximally exploit that architecture

SISSA
ICTP

MHPC
Master in High Performance Computing

## What is available ?

- Linear Algebra: BLAS/LAPACK/SCALAPACK
- FFT:
  - FFTW
- Finite elements: DEALII / PETSC
- Good starting point: www.netlib.org

MHPC
Master in High Performance Computing

## Why use performance libraries ?

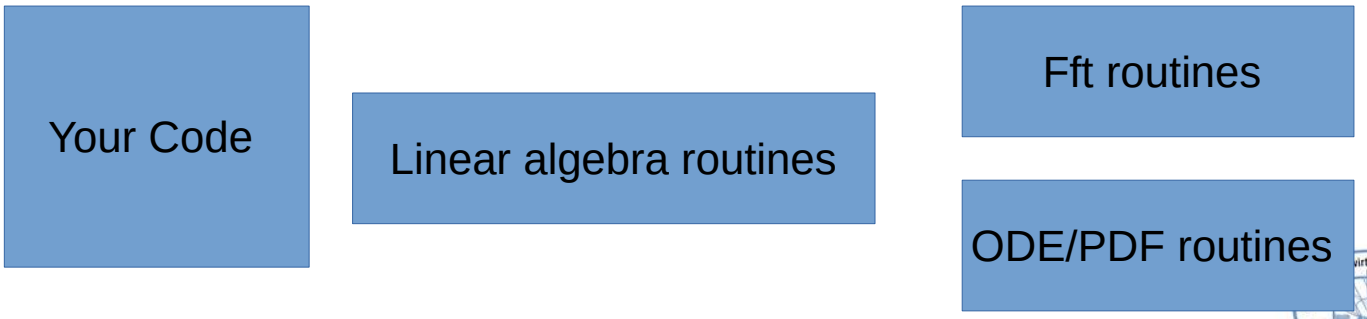
- Performance libraries are designed to use the CPU in the most efficient way, which is not necessarily the most straightforward way.
- It is normally best to use the libraries supplied by or recommended by the CPU vendor
- On modern hardware they are hugely important, as they most efficiently exploit caches, special instructions and parallelism

SISSA
ICTP

MHPC
Master in High Performance Computing

## Any other reason apart from optimization ?

- Usage of libraries
  - Make coding easier. Complicated math operations can be used from existing routines
  - Increase portability of code as standard (and well optimized) libraries exist for ALL computing platforms.
- Lego approach: build your own code using available bricks..




SISSA
ICTP

MHPC
Master in High Performance Computing

## Standardization (BLAS example)

- Subroutines have a standardized layout
- BLAS is documented in the source code
- Man pages exist
- Vendor supplied docs
- Different BLAS implementations have the same calling sequence

```
      SUBROUTINE DGEMM ( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB,
     $                   BETA, C, LDC )
*     .. SCALAR ARGUMENTS ..
      CHARACTER*1        TRANSA, TRANSB
      INTEGER            M, N, K, LDA, LDB, LDC
      DOUBLE PRECISION   ALPHA, BETA
*     .. ARRAY ARGUMENTS ..
      DOUBLE PRECISION   A( LDA, * ), B( LDB, * ), C( LDC, * )
*     ..
*
*  PURPOSE
*  =======
*
*  DGEMM  PERFORMS ONE OF THE MATRIX-MATRIX OPERATIONS
*
*     C := ALPHA*OP( A )*OP( B ) + BETA*C,
*
*  WHERE  OP( X ) IS ONE OF
*
*     OP( X ) = X   OR   OP( X ) = X',
*
*  ALPHA AND BETA ARE SCALARS, AND A, B AND C ARE MATRICES, WITH OP( A )
*  AN M BY K MATRIX,  OP( B )  A  K BY N MATRIX AND  C AN M BY N MATRIX.
*
*  PARAMETERS
*  ==========
*
*  TRANSA - CHARACTER*1.
*           ON ENTRY, TRANSA SPECIFIES THE FORM OF OP( A ) TO BE USED IN
*           THE MATRIX MULTIPLICATION AS FOLLOWS:
*
*              TRANSA = 'N' OR 'N',  OP( A ) = A.
*
*              TRANSA = 'T' OR 'T',  OP( A ) = A'.
*
*              TRANSA = 'C' OR 'C',  OP( A ) = A'.
*
*           UNCHANGED ON EXIT.
*
*  TRANSB - CHARACTER*1.
*           ON ENTRY, TRANSB SPECIFIES THE FORM OF OP( B ) TO BE USED IN
*           THE MATRIX MULTIPLICATION AS FOLLOWS:
*
*              TRANSB = 'N' OR 'N',  OP( B ) = B.
*
*              TRANSB = 'T' OR 'T',  OP( B ) = B'.
```

SISSA (CTP)

MHPC
Master in High Performance Computing

# Highly optimize library BLAS/LAPACK libraries available

- Intel Math Kernel Library (MKL): BLAS/LAPACK (+FFT) routines modified for best performance on IA32/x86, x86_64/AMD64/EM64t, and IA64 based machines.
- AMD Math Core Library (ACML): BLAS LAPACK (+FFT) routines modified for best performance on AMD x86 and x86_64 based machines
- Automatically Tuned Linear Algebra Software (ATLAS): BLAS and LAPACK routines that use empirical tests to tune machine specific parameters.
- openBLAS implementation with special optimization for modern (x86/x86_64) CPU architectures
- Other libraries: PLASMA-MAGMA (see future lectures..)

SISSA ICTP

MHPC
Master in High Performance Computing

## Linking your code to libraries

- Linker flags: -L/some/other/dir -lm -> search for libm.so/libm.a also in /some/dir
- Order matters. Ex.: LAPACK uses BLAS => -L/usr/local/lib -llapack -lblas
- ATLAS is written C with f77 wrappers: => -L/opt/atlas/lib -lf77blas -latlas
- MKL uses “collections”. Using -lmkl links: libmkl_intel_lp64.so, libmkl_intel_thread.so libmkl_core.so => check with “ldd”

  check: https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor
- Check LD_LIBRARY_PATH with shared libs

SISSA ICTP

**Some of them are multithreaded libraries !**

- Precompiled by means of openMP and/or any other threaded library
- ATLAS: fixed number of threads compiled in
  - From FAQ: No. The maximum number of threads to use is determined at compile time. ATLAS will never use more than this, but may use less if the problem sizes are too small to get speedup from the additional parallelism.
- OpenBLAS: threading with OpenMP
- MKL: Threading with OpenMP
  - See documentation for details (always RTFM!).

MHPC
Master in High Performance Computing

## How to compile/ link a multithreaded library ? MKL

- Default linking sequence for ifort with Intel threads => add -openmp to linker/compiler command
- Serial: -lmkl_intel_lp64 -lmkl_sequential -lmkl_core
- Threaded
- Again check
- https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor
- See documentation for details (always RTFM!).

SISSA ICTP

MHPC
Master in High Performance Computing

## And now...

- Stream : see D7-material/stream
- MPI benchmarks: see D7-material/mpi
- Nodeperf: see D7-material/nodeperf
- HPL

SISSA
ICTP