Getting Performance from OpenMP Programs on NUMA Architectures

Christian Terboven, RWTH Aachen University
terboven@itc.rwth-aachen.de

EU H2020 Centre of Excellence (CoE)

1 October 2015 – 31 March 2018

Grant Agreement No 676553

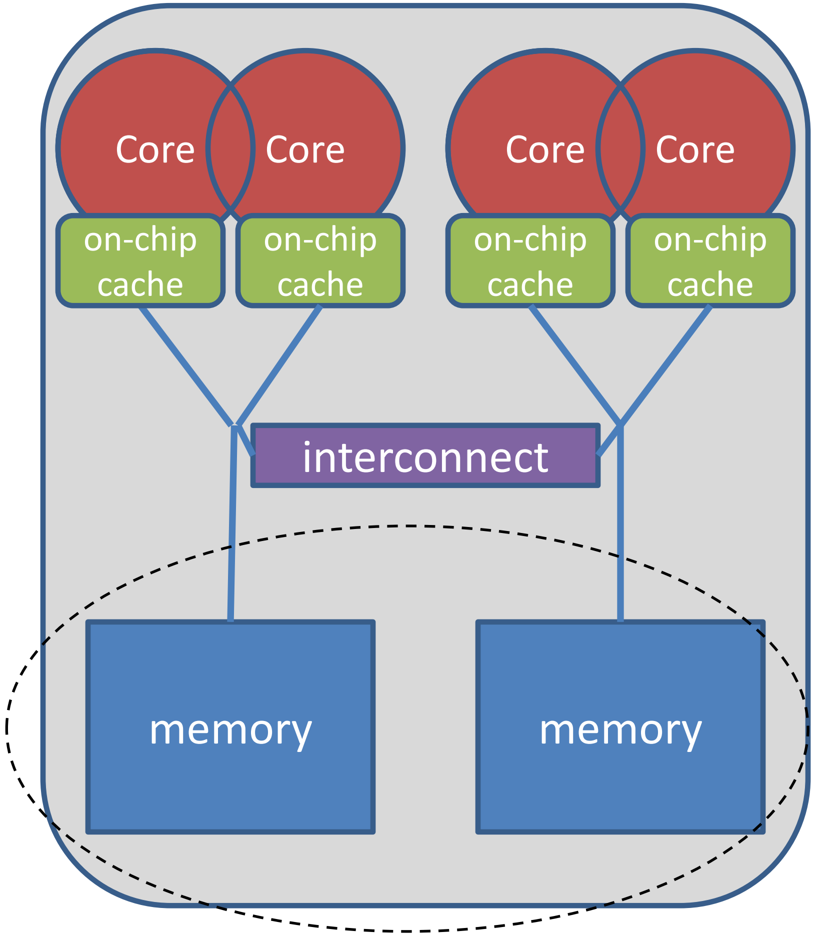
OpenMP and Performance

- Among the most obscure things that can negatively impact performance of OpenMP programs are cc-NUMA effects

- ***These are not restricted to OpenMP***
  - But they most show up because you used OpenMP
  - In any case they are important enough to be covered here

What is cc-NUMA?

- Set of processors is organized inside a locality domain with a locally connected memory
  - The memory of all locality domains is accessible over a shared virtual address space
  - Other locality domains are access over a interconnect, the local domain can be accessed very efficiently without resorting to a network of any kind



Virtual address space

Advantages & Disadvantages

- Advantages
  - Scalable in terms of memory bandwidth
  - "Arbitrarily" large numbers of processors: There exist systems with over 1024 processors
- Disadvantages
  - Efficient programming requires precautions with respect to local and remote memory, although all processors share one address space
  - Cache coherence is hard and expensive in implementation
    - e.g. recent writes need invalidation and may consume a lot of the available bandwidth

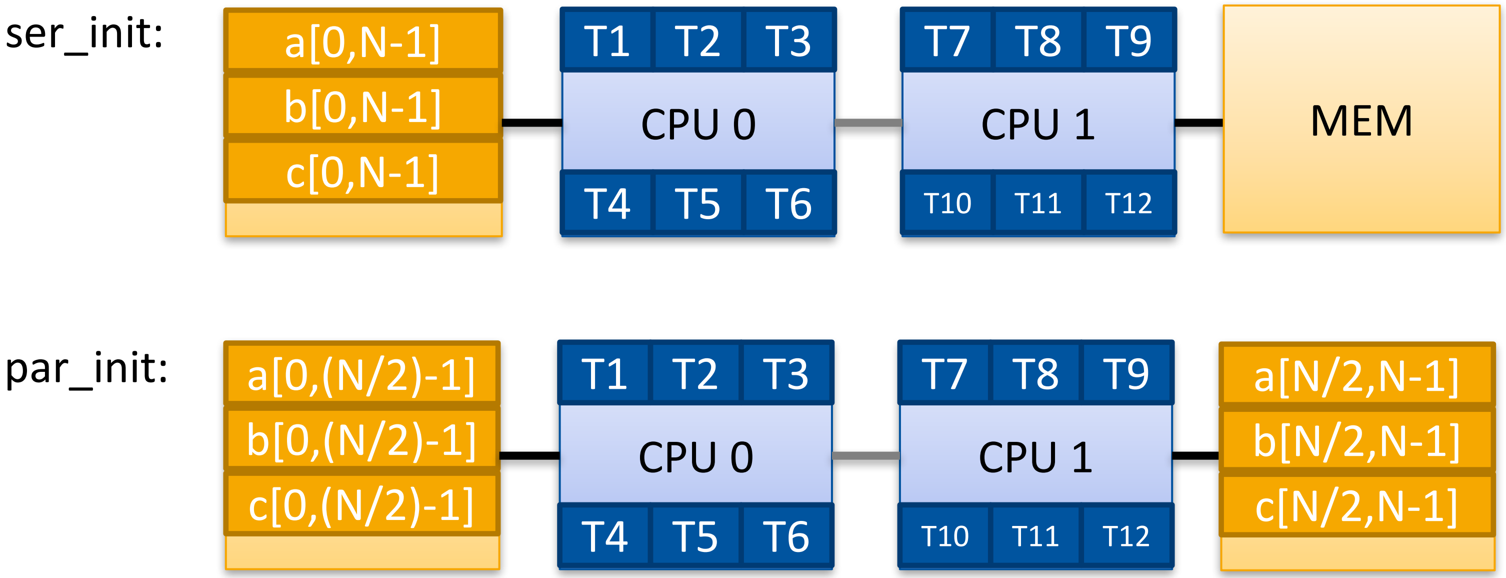
Query your System

- You should have a basic understanding of the system topology. You could use one of the following options on a target machine:
  - numactl tool to control the Linux NUMA policy
    - `numactl --hardware`
    - Delivers compact information about NUA nodes and the associated processor ID
  - Intel MPI‘s `cpuinfo` tool
    - `cpuinfo`
    - Delivers information about the number of sockets (= packages) and the mapping of processor IDs to CPU cores used by the OS
  - hwlocs‘ `hwloc-ls` tool (comes with Open-MPI)
    - `hwloc-ls`
    - Displays a (graphical) representation of the system topology, separated into NUMA nodes, along with the mapping of processor IDs to CPU cores used by the OS and additional information on caches

To be NUMA or not to be

- Stream example with and without NUMA-aware data placement
  - 2 socket system with Xeon X5675 processors, 12 OpenMP threads

	copy	scale	add	triad
ser_init	18.8 GB/s	18.5 GB/s	18.1 GB/s	18.2 GB/s
par_init	41.3 GB/s	39.3 GB/s	40.3 GB/s	40.4 GB/s

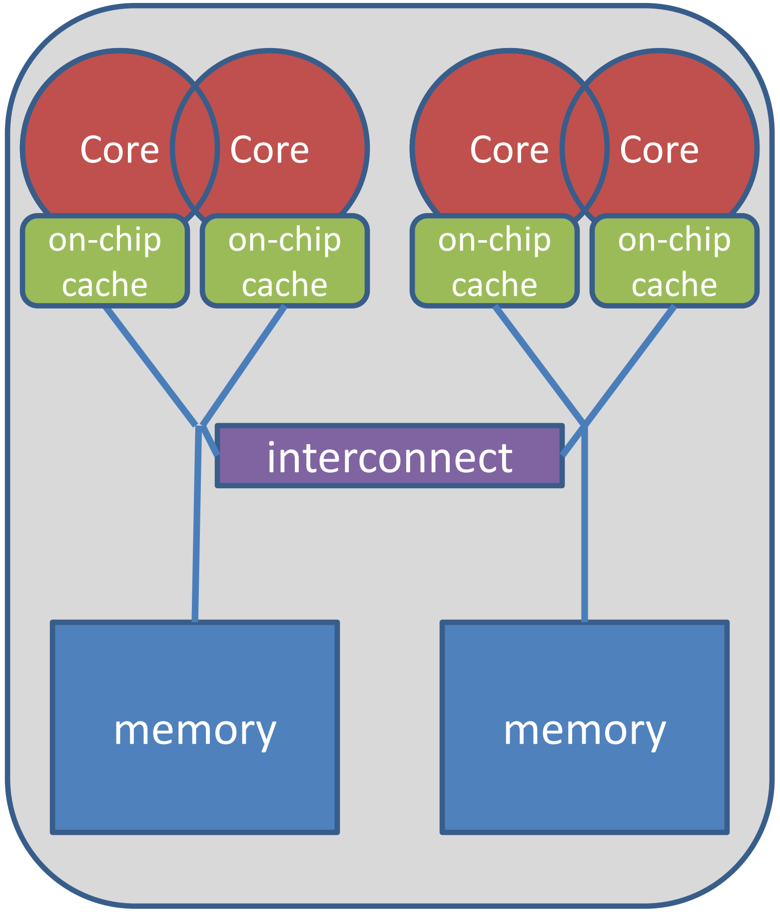


Data Placement?

How To Distribute The Data ?

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

```
for (int i = 0; i < N; i++) {
   A[i] = 0.0;
}
```



Serial Data Placement

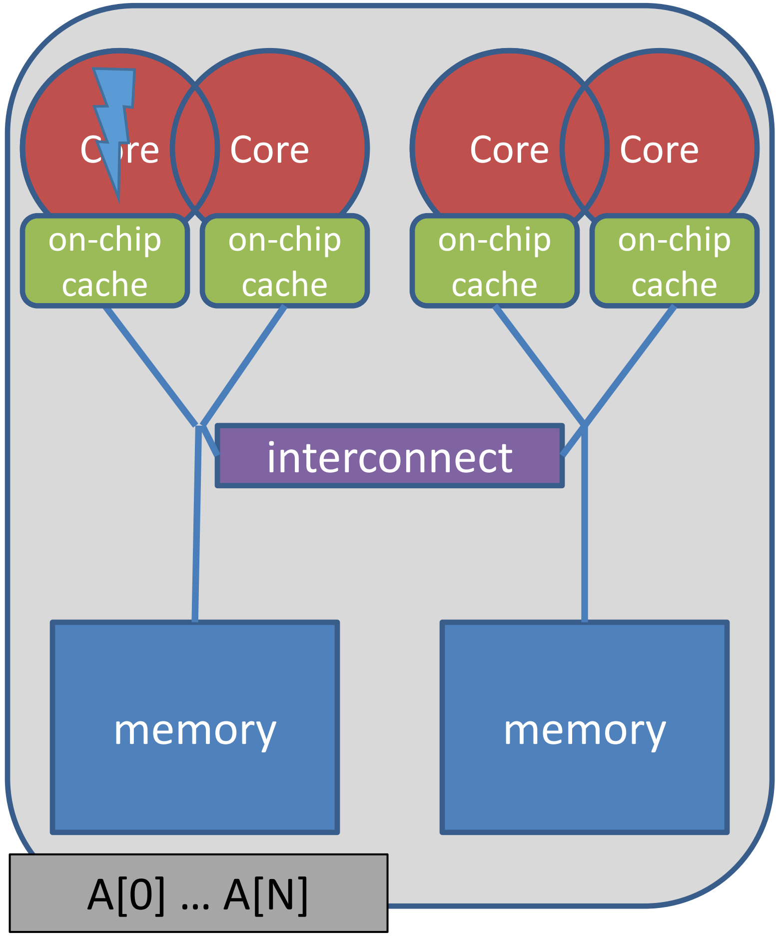
- Serial code: all array elements are allocated in the memory of the NUMA node closest to the core executing the initializer thread (first touch)

```
double* A;

A = (double*)
    malloc(N * sizeof(double));




for (int i = 0; i < N; i++) {
    A[i] = 0.0;
}
```



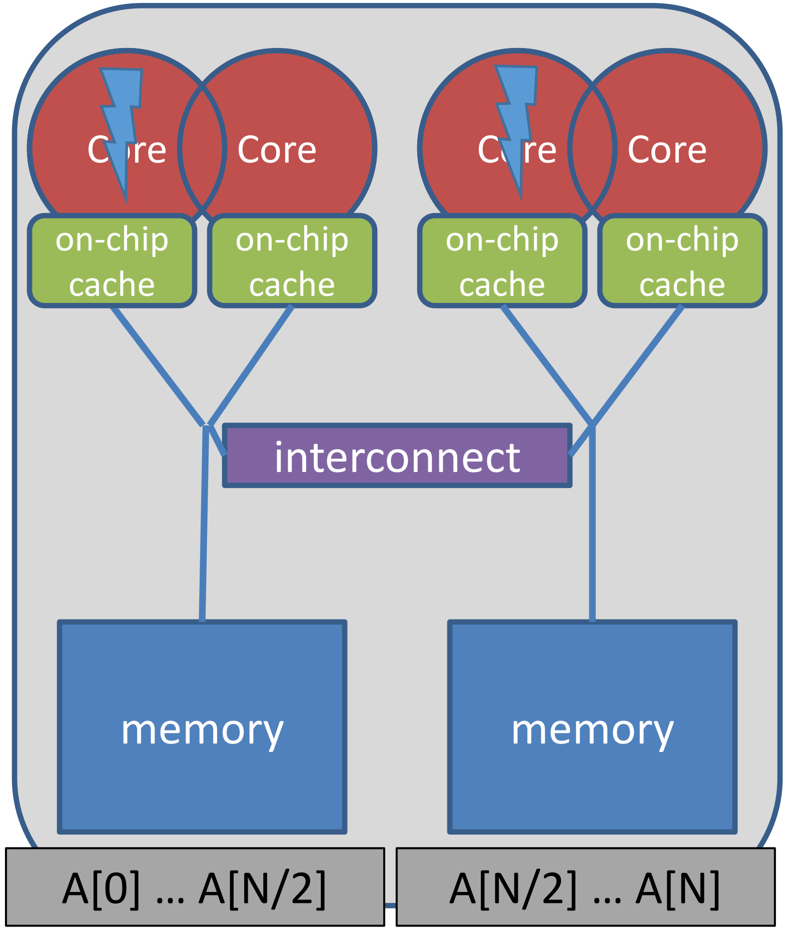
First Touch Data Placement

- Serial code: all array elements are allocated in the memory of the NUMA node closest to the core executing the initializer thread (first touch)

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



#pragma omp parallel for
for (int i = 0; i < N; i++) {
   A[i] = 0.0;
}
```



Decide for Binding Strategy

- Selecting the „right" binding strategy depends not only on the topology, but also on the characteristics of your application
  - Putting threads far apart, i.e., on different sockets
    - May improve the aggregated memory bandwidth available to your application
    - May improve the combined cache size available to your application
    - May decrease performance of synchronization constructs
  - Putting threads close together, i.e., on two adjacent cores that possibly share some caches
    - May improve performance of synchronization constructs
    - May decrease the available memory bandwidth and cache size

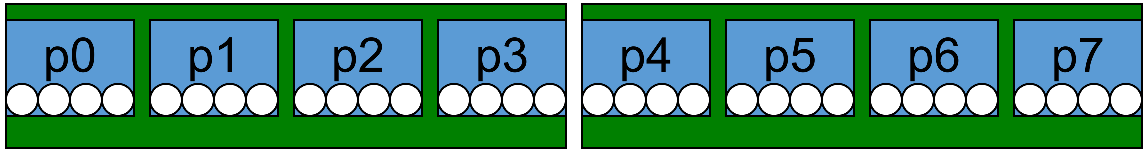
- If you are unsure, just try a few options and then select the best one.

OpenMP 4.0: Places + Policies

- Define OpenMP places
  - set of OpenMP threads running on one or more processors
  - can be defined by the user, i.e., `OMP_PLACES=cores`
- Define a set of OpenMP thread affinity policies
  - SPREAD: spread OpenMP threads evenly among the places, partition the place list
  - CLOSE: pack OpenMP threads near master thread
  - MASTER: collocate OpenMP threads with master thread
- Goals
  - user has a way to specify where to execute OpenMP threads for locality between OpenMP threads / less false sharing / memory bandwidth

OMP_PLACES env. variable

- Assume the following machine:



  - 2 sockets, 4 cores per socket, 4 hyper-threads per core

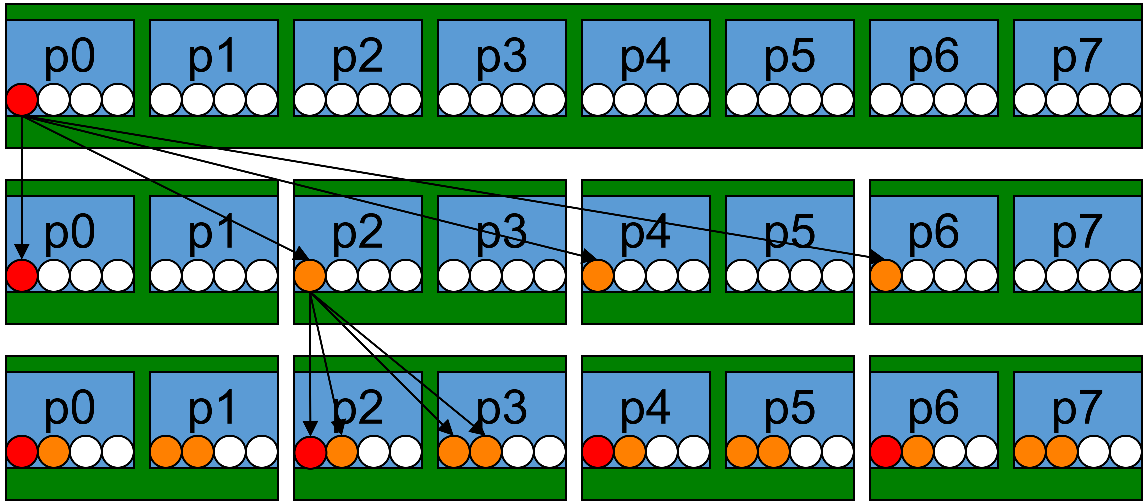
- Abstract names for OMP_PLACES:
  - threads: Each place corresponds to a single hardware thread on the target machine
  - cores: Each place corresponds to a single core (having one or more hardware threads) on the target machine
  - sockets: Each place corresponds to a single socket (consisting of one or more cores) on the target machine

OpenMP 4.0: Places + Policies

- Example's Objective:
  - separate cores for outer loop and near cores for inner loop
- Outer Parallel Region: proc_bind(spread), Inner: proc_bind(close)
  - spread creates partition, close binds threads within respective partition

```
OMP_PLACES=(0,1,2,3), (4,5,6,7), ... = (0-4):4:8    = cores
#pragma omp parallel proc_bind(spread) num_threads(4)
#pragma omp parallel proc_bind(close) num_threads(4)
```

- Example
  - initial
  - spread 4
  - close 4



NUMA Strategies: Overview

- First Touch: Modern operating systems (i.e., Linux >= 2.4) determine the physical location of a memory page during the first page fault, when the page is first „touched“, and put it close to the CPU that causes the page fault
- Everything under control?
- In principle Yes, but only if
  - threads can be bound explicitly,
  - data can be placed well by first-touch, or can be migrated,
- What if the data access pattern changes over time?

- Explicit Migration: Selected regions of memory (pages) are moved from one NUMA node to another via explicit OS syscall

- Automatic Migration: No support for this in current operating systems

User Control of Memory Affinity

- Explicit NUMA-aware memory allocation:
  - By carefully touching data by the thread which later uses it
  - By changing the default memory allocation strategy with `numactl`
  - By explicit migration of memory pages
    - Linux: `move_pages()`
      - Include `<numaif.h>` header, link with `-lnuma`
        long move_pages(int ***pid*****, unsigned long count, void **** ***pages*****, const int *** ***nodes*****, int *** ***status*****, int** ***flags*****);**

- Example: using numactl to distribute pages round-robin:
  - `numactl –interleave=all ./a.out`
- Example: Run on node 0 with memory allocated on nodes 0 and 1
  - `numactl --cpubind=0 --membind=0,1 ./a.out`

Performance Optimisation and Productivity

A Centre of Excellence in Computing Applications

Contact:

https://www.pop-coe.eu

mailto:pop@bsc.es

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676553.