HPC2N, UmU

Thread binding for OpenMP

Joachim Hein
LUNARC & Centre of Mathematical Sciences
Lund University

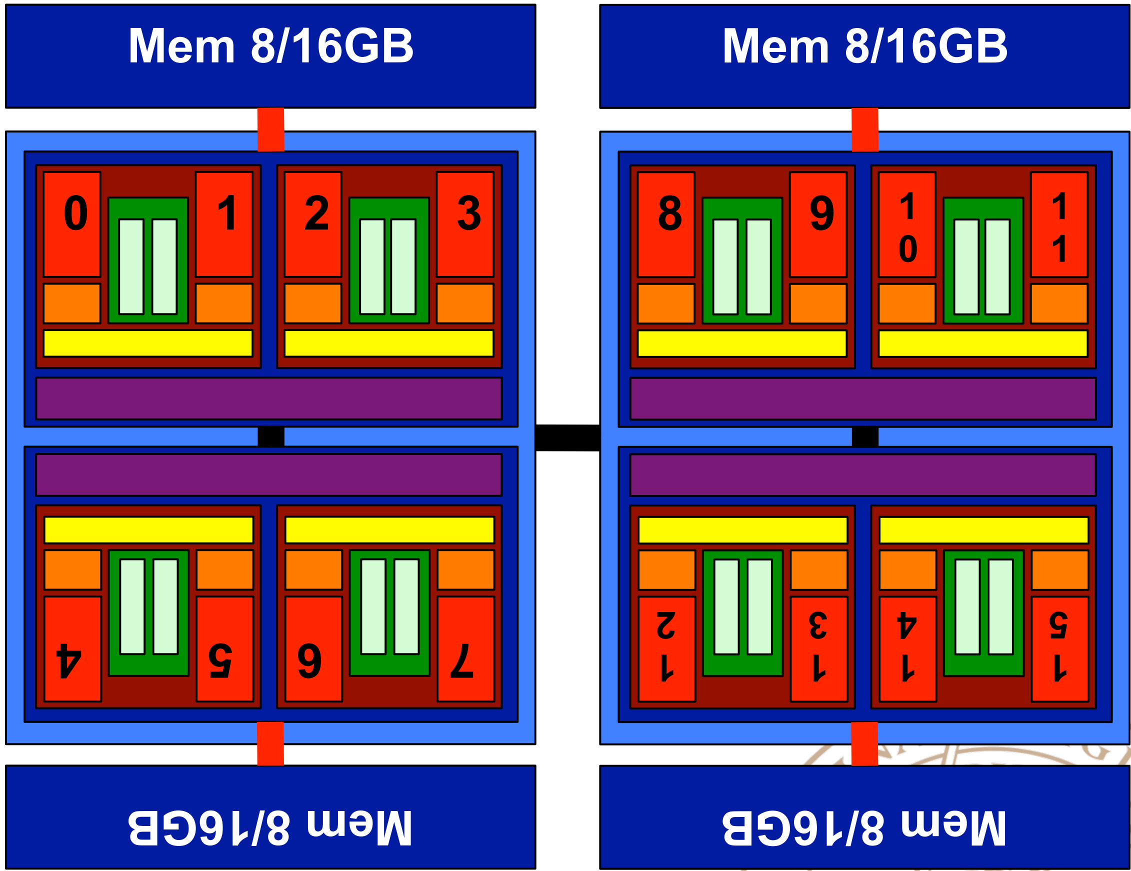
Pedro Ojeda May
HPC2N
Umeå University

OS Thread-scheduling

- Operating system schedules processes & threads
  - Running vs sleeping
  - Assignment to physical hardware
- Processes and threads typically scheduled freely
  - OS moves long running processes with time
  - Utilise as many cores as possible at all time
  - Good: web- or transactionserver, Interactive multiuser
- Can create problems for parallel HPC applications
  - Loss of memory locality
  - Multiple compute threads on same core

Structure of 16-way AMD Interlagos node
Example for a CC-NUMA architecture

- 2 Processors
- 4 dual module groups
- Memory attached to these groups (CC-NUMA architecture)
- Hypertransport
- 16 cores



Thread binding

- Threads allocate memory locally if available
- Thread binding
  - Bind threads to hyper-threads, cores or groups thereof
  - Avoid multiple threads on a single core/hyper-thread
  - Threads are not moved away from “their” memory
  - Allows optimisation of inner node communication

- Up to OpenMP 3.1, thread binding was compiler dependent
  - OpenMP 4.0 provides standardised binding features

- Watch out for hyper-threading

Mapping of threads on the hardware

- Mapping of threads onto hardware affects performance
- Effect to consider
  - Shared caches facilitating fast data exchange
  - Multiple memory busses / memory controllers
  - Shared functional units (e.g. FPU in AMD Interlagos)
- Particular important for under populated nodes

OpenMP binding a two stage process

You need to define

- Places
  - Where should the threads run
  - Examples: Hyper-thread, Core, Socket (Processor)
  - Groups of the above
- Affinity policy
  - How threads get distributed on the “Places"

OpenMP Places

- Controlled by environment variable: `OMP_PLACES`
  – Not changeable at runtime

- Defines hardware resources to execute OpenMP threads
  – Can execute more than one OpenMP thread

- Easiest to use abstract names, choices:

```
export OMP_PLACES=threads      (Hyper-thread)
export OMP_PLACES=cores        (physical Core)
export OMP_PLACES=sockets      (Processor)
```

Specifying an OpenMP place

- Specify a place as a list of positive numbers
  - Labeling smallest execution unit exposed by environment
  - Typically hardware thread
- Specifying places
  - You can define a place as comma separated list in {}
    - E.g: `{0,4,8,12}` – system can choose from 4 units
  - Define a place as interval with a colon:
    - E.g: `{4:5}` – this corresponds to `{4,5,6,7,8}`
    - The 5 give the number of units, not the end!
  - You can define a stride:
    - E.g: `{5:4:3}` – this corresponds to `{5,8,11,14}`

The OMP_PLACES variable as a list

- Comma separated list of places

  ```
  export OMP_PLACES="{0,1,2,3},{4:4},{8:4}"
  ```

- You can specify (strided) intervals for the place list

  ```
  OMP_PLACES="{0,1,2,3}:3:5"
  ```

  corresponds to:

  ```
  OMP_PLACES="{0,1,2,3},{5,6,7,8},{10,11,12,13}"
  ```

Thread binding: Affinity policy

- Affinity policy is controlled by the environment variable `OMP_PROC_BIND`

`OMP_PROC_BIND`	effect
`false`	no binding
`true`	binding enabled
`master`	bind to same place as master
`close`	bind subsequent threads to subsequent places, round-robin when all used
`spread`	bind threads as far away as possible

- You can provide a list containing `master`, `close` and `spread`, that will be used for nested parallel regions

Declaring binding on the parallel region OpenMP 4.0

- One can declare binding in your source for a parallel region
- Binding (`OMP_PROC_BIND`) must not be `false`
- Fortran:

```
!$omp parallel proc_bind(policy)
```

- C

```
#pragma omp parallel proc_bind(policy)
```

- Valid values for *`policy`* are:

```
master
close
spread
```

Effect of Binding
Calulating a vector norm

- icc 17.0.1
- Haswell node:
  2 Intel E5-2650 v3
- Benchmark:
  - Initialising the vector and calculating the norm
  - Separate loops, cache inefficient
- Small problem: Communication bound
  - Staying inside a processor most efficient
- Large problem: Streams bound
  - Utilising both processors more efficient

Treads	Size	None	Close	Spread
10	4000	4.7µs	2.5µs	4.7µs
20	4000	6.2µs	5.0µs	5.1µs
10	400M	96ms	179ms	94ms
20	400M	90ms	90ms	90ms

Summary

- Explained the need for thread binding
- Since OpenMP 4.0: binding part of the OpenMP standard
  - Previously compiler dependent
- Demonstrated effect of binding on a simple benchmark