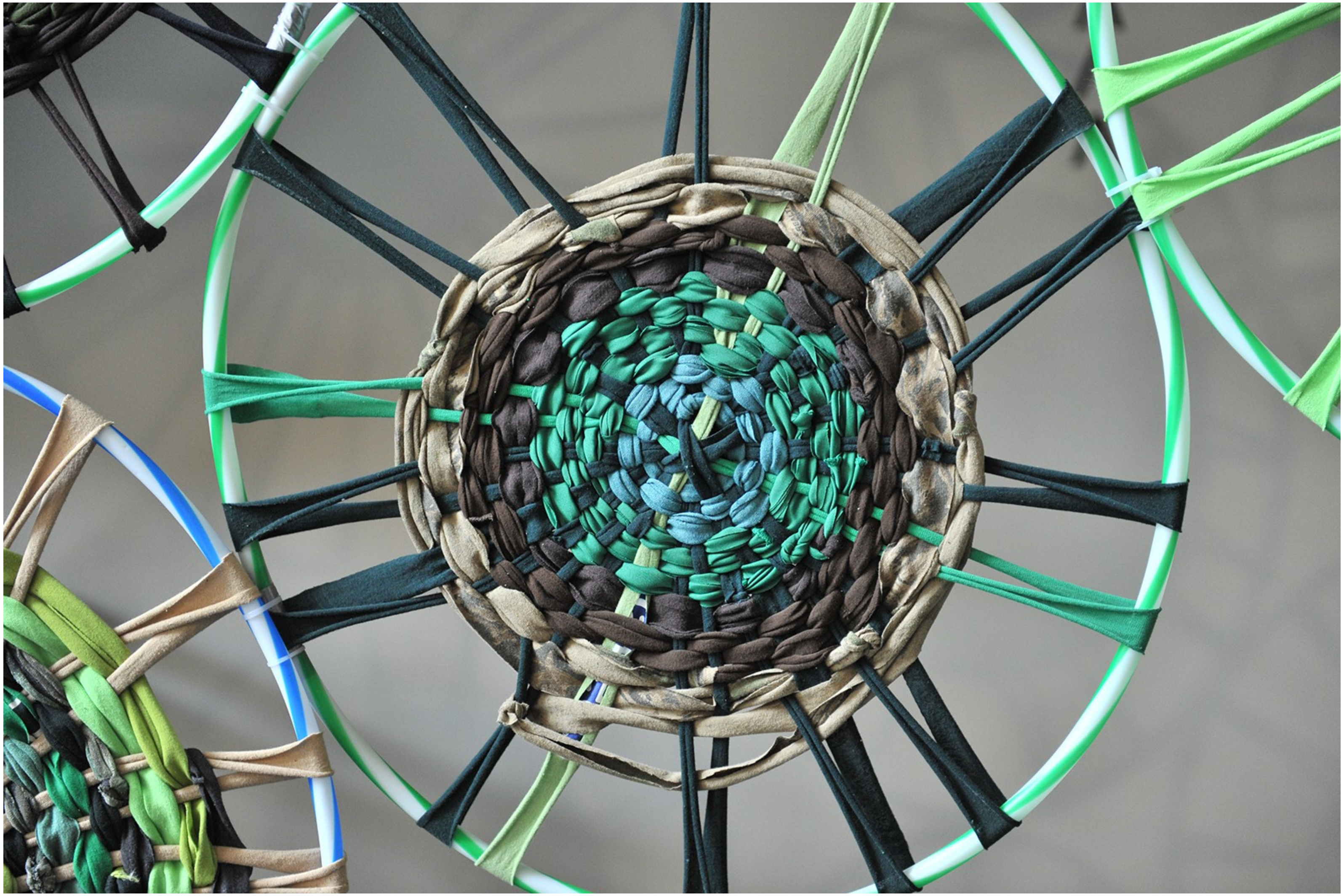
OpenMP 4



University of BRISTOL

Overview

- New features in OpenMP v4:
  - CPU affinity.
  - Compute devices.
  - Task groups.
  - SIMD directive.

Not everything we present is focussed exclusively on the assignment. Today is a mix.

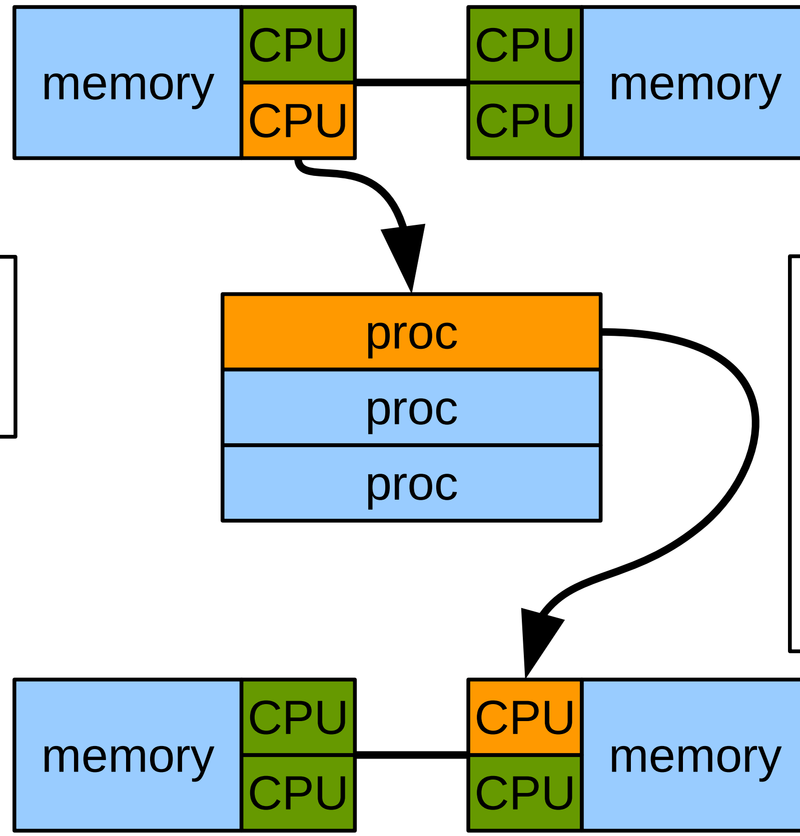
University of BRISTOL

New in OpenMP v4:

CPU Affinity

What is CPU Affinity?

Affinity: which process is running on which processor core?



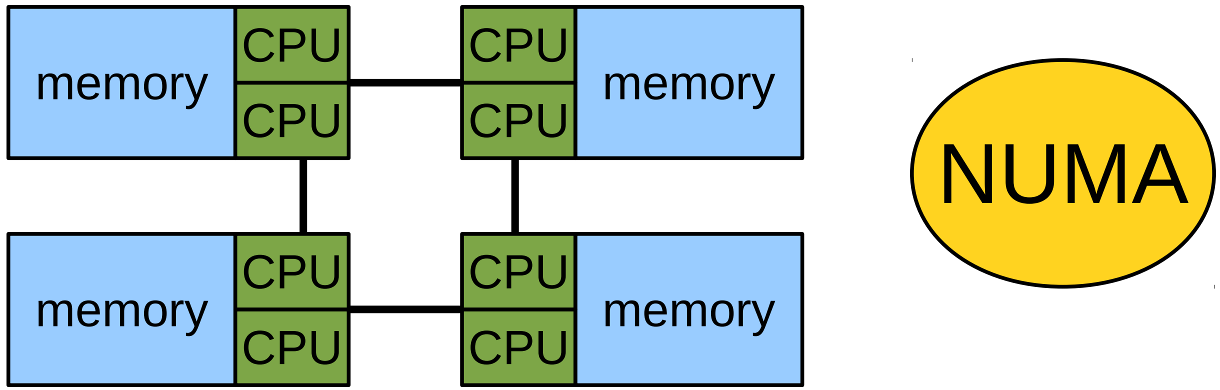
- Linux kernel responsible for scheduling many processes.
- Left to its own devices, processes may be moved about without much NUMA awareness.

Specifying CPU Affinity

- Previously...
- We had OpenMP runtime specific methods for controlling CPU affinity (via env vars), e.g.
  - Intel: `export KMP_AFFINITY={compact/scatter}`
  - GNU: `export GOMP_CPU_AFFINTY="0 3 1-2 4-15:2"`
- And command line tools, such as:
  - `taskset -c 0-7 ./myprog.exe`
  - `numactl --cpunodebind=0 –membind=0 ./myprog.exe`

University of BRISTOL

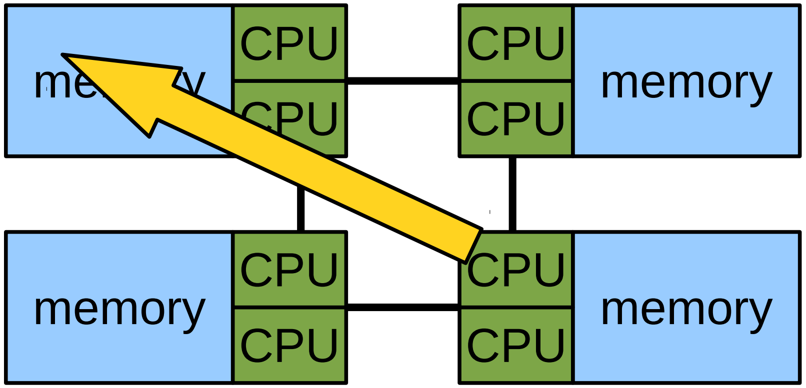
Why? What's the Big Deal? Non Uniform Memory Access



- A very common architecture.
- Slower access to more distant memory can be in tension with, say, attempts to redistribute loop iterations for load balance.
- Knowledge of O/S memory allocation policies and thread affinity controls can diffuse some of those tensions.

University of BRISTOL

NUMA



- O/S uses 'first touch' policy to control placement of items across the banks of memory.
- So, be mindful of how you initialise your data structures, i.e. the 'first touch'.
- And then how you subsequently access them in a parallel region.

University of BRISTOL

Specifying CPU Affinity

- Now..

e.g. h/w hyperthreads

- `export OMP_NUM_THREADS=16`
- `export OMP_PLACES=threads|cores|sockets`
  - NB the above does not set any affinity,

    but provides a context for the binding policy.
- `export OMP_PROC_BIND=master|close|spread`
- `#pragma omp parallel proc_bind(spread) num_threads(4)`
- A more portable solution.

Workers share resource with master

- (Can assign different affinities to different levels in a nested parallel region.)

University of BRISTOL

New in OpenMP v4:

Compute Devices

Compute Devices


```
/* determine the number of accelerators available */
ndevices = omp_get_num_devices();
printf("Number of accelerators = %d\n", ndevices);

/* report the number of threads available on each device */
for(i=0; i<ndevices; i++)
  {
#pragma omp target device(i)       /* set the target */
#pragma omp parallel               /* open a parallel region */
    {
#pragma omp master                 /* only the master thread need query */
      {
        nthreads = omp_get_num_threads();
        printf("Device number = %d\tNumber of threads = %d\n", i, nthreads);
      }
    }
```


University of BRISTOL

Compute Devices


```
/* determine the number of accelerators available */
ndevices = omp_get_num_devices();
printf("Number of accelerators = %d\n", ndevices);

/* control the distibution of computational work */
for(i=0; i<ndevices; i++)
  {
#pragma omp target device(i)        /* set the target */
#pragma omp teams num_teams(2) num_threads (32) /* make a league of teams */
    {
#pragma omp distribute                      /* distribute the loops over the teams */
      for(ii=0; ii<N; ii++) {}
    }
```

You can create teams of threads, which share memory and may work well with the design of the device hardware.

..And Transferring Data?

```
float *x = (float*) malloc(n * sizeof(float));
float *y = (float*) malloc(n * sizeof(float));

#pragma omp target device(0) map(to:x) map(tofrom:y)
 {
#pragma omp parallel for
   for (ii=0; ii < n; ii++) {
     y[ii] = a*x[ii] + y[ii];
   }
 }
```

Notice the use of the word '***map***' and compare to, say, OpenACC:

```
#pragma acc kernels loop copyin(x[0:n]) copy(y[0:n]) independent
 for (ii=0; ii<n; ii++) {
   y[ii] = a*x[ii] + y[ii];
 }
```

This is appropriate, because.. a number of vendors will soon offer devices with on package memory and direct access to main memory.

NVIDIA

University of BRISTOL

..And Transferring Data?

- Types of mapping:
  - **`to`**: existing host variables are copied to a corresponding variable on the target before the actions in the code block.
  - **`from`**: target variables copied back to a corresponding variable in the host after the actions of the code block.
  - **`tofrom`**: Both to and from.
  - **`alloc`**: Neither to nor from. Ensure the variable exists on the target but it has no relation to a host variable.

New in OpenMP v4:

Task Groups
SIMD Directive

Task Groups

Can cancel groups of tasks, e.g.:

serial

```
for(ii=0, ii < N; ii++) {
  …
  if(....) {
   break;
  }
...
}
```

parallel

```
#pragma omp parallel for
for(ii=0, ii < N; ii++) {
  …
  if(....) {
   #pragma omp cancel for
  }
...
}
```

- Can also group tasks inside a region: **`#pragma omp taskgroup`**.
- Growing expressiveness in the 'language'.

University of BRISTOL

SIMD Directive

- New directive to indicate that, e.g. a work sharing loop can be executed using SIMD lanes:
  - #pragma omp parallel for simd
- However, the compiler will also make a decision about whether it can vectorise a loop. So this is a useful compiler hint, but not a magic wand!
  - #pragma omp simd
    - Flag a loop can be executed using SIMD lanes
  - #pragma omp declare simd
    - Flags a function that can be called from a SIMD loop

Other Features

- User-defined reductions:
  - More control over this efficient construct.
- For affinity:
  - OMP_DISPLAY_ENV=TRUE|FALSE|VERBOSE
  - OpenMP will print out at runtime how it has interpreted your specification.

University of BRISTOL

OpenMP v4 on BC3

- OpenMP Examples #7
- You'll need to use v16 of the Intel compiler, or above:
  - `module add languages/intel-compiler-16`
  - e.g. can use `#pragma omp simd`
- We have two nodes on BC3 equiped with KNC generation Xeon-Phi cards:
  - `qsub -q testq -I`

    `-l nodes=1:ppn=16:xeon-phi`

University of BRISTOL

Summary

- OpenMP v4 adds support for:
  - Specifying CPU affinity, in a portable and nuanced way.
  - Use of compute devices, such as GPUs and other accelerators.
  - Compiler hints specifying SIMD operations.
  - Task groups, user-defined reductions etc.