Advanced OpenMP

ICHEC

Irish Centre for High-End Computing

OpenMP 4.0

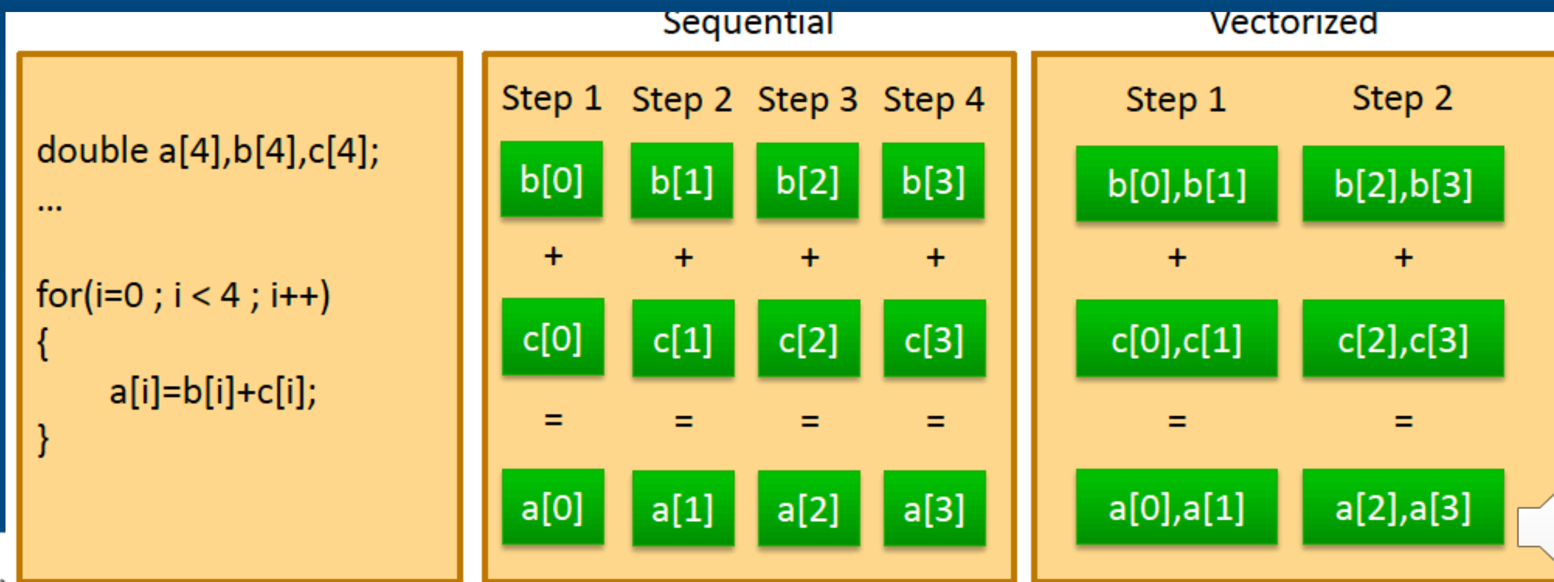
OpenMP 4.0 released in July 2013

- SIMD directives
- Extended support for Thread Affinity
- New User-defined Reductions
- Error Handling
- Accelerators support
- Support for Fortran 2003

ICHEC
Irish Centre for High-End Computing

Vectorisation Support …(1)

- Vectorisation: Execute a single instruction on multiple data objects in parallel within a single CPU core
- Many compilers exploit vector parallelism which is limited due to dependencies, inner loops, function calls etc.
- More architectures support longer vector length
- Simd: Single Instruction, multiple data



ICHEC
Irish Centre for High-End Computing

Vectorisation Support ...(2)

- Divide loop iterations into chunks that fit a simd vector register (vector length is hardware dependent)
- Multiple iterations of the loop can be executed concurrently

C/C++:
```
#pragma omp simd [clauses]
   for – loops
```

Fortran:
```
!$omp simd [clauses]
   do – loops
!$omp end simd
```

- Clauses: safelen, linear, aligned, private, lastprivate, reduction, collapse
  - safelen (length): limits the number of iterations in a SIMD chunk
  - linear (list): declares a number of list items to be private to a SIMD lane
  - aligned (list): declares a number of items to be aligned to some number of bytes

ICHEC
Irish Centre for High-End Computing

Vectorisation Support …(3)

- Parallelise and vectorise a loop nest, distribute loop iterations across thread team, subdivide loop chucks to fit a simd register

C/C++:
```
#pragma omp parallel for simd [clauses]
  for – loops
```

Fortran:
```
!$omp parallel do simd [clauses]
  do – loops
!$omp end do simd
```

- Simd function vectorisation: for elemental functions that are called from within a loop, so compilers can vectorise the function.

```
#pragma omp declare simd [clauses]
  function
```

```
!$omp declare simd [clauses]
  function
!$omp end simd
```

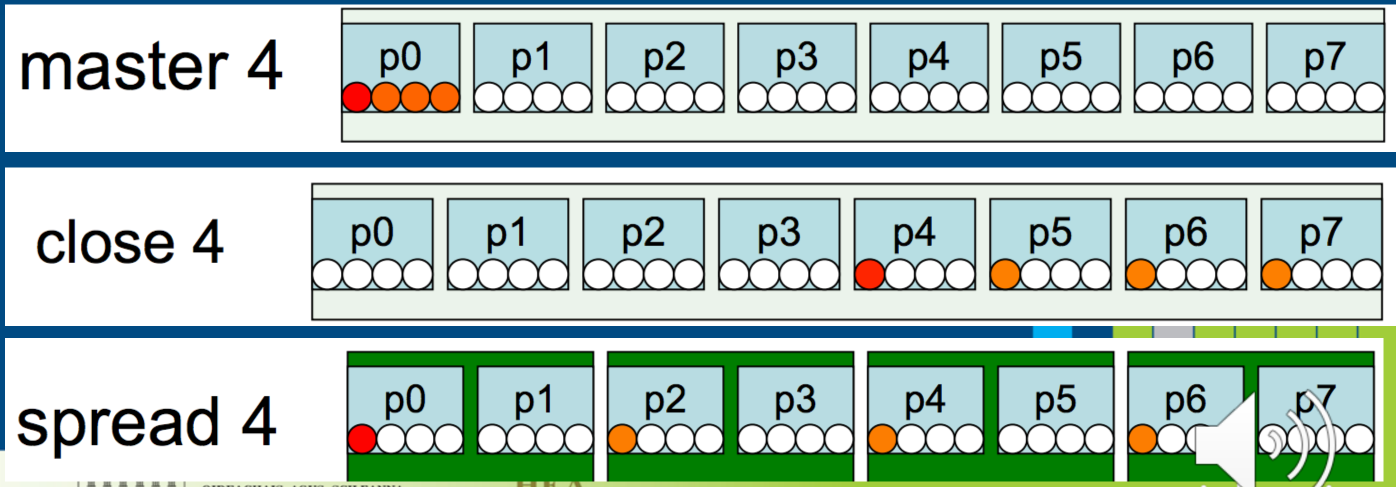
- Clauses: simdlen, linear, aligned, uniform, reduction, inbranch, notinbranch

ICHEC Irish Centre for High-End Computing

Thread Affinity

- Allow users to have finer control how and where OpenMP threads are placed
- OMP_PLACES: Binds the OpenMP threads to the places; threads: each place corresponds to a single hardware thread, cores: corresponds to a single core (having one or more hardware threads), sockets: corresponds to a single socket (consisting of one or more cores)
- OMP_PROC_BIND / proc_bind(master|close|spread): specifies how threads are bound to OpenMP places; master: threads to the same place as the master thread, close: threads close to the place of the master thread, spread: spread threads across the machine

OMP_PLACES=threads



ICHEC
Irish Centre for High-End Computing

User-defined Reduction …(1)

- Use your own reduction operation on your own type

C/C++:

```
#pragma omp declare reduction(reduction-identifier: typename-list : combiner) [initializer-clause]
```

Fortran:

```
!$omp declare reduction(reduction-identifier: type-list : combiner) [initializer-clause]
```

- Reduction-identifier: gives a name to the operator
- Typename-list: A list of types to which it applies
- Combiner: specifies how to combine values
- Initializer-clause: specifies how to initialise private elements of each thread

ICHEC
Irish Centre for High-End Computing

User-defined Reduction …(2)

```
#pragma omp declare reduction (merge : std::vector<int> :
omp_out.insert(omp_out.end(), omp_in.begin(), omp_in.end()))
```

```
void schedule (std::vector<int> &v, std::vector<int> &filtered)
{
#pragma omp parallel for reduction (merge : filtered)
   for (std:vector<int>::iterator it = v.begin(); it < v.end();
it++)
       if ( filter(*it)) filtered.push_back(*it);
}
```

- omp_out refers to private copy that holds combined value
- omp_in refers to the other private copy

ICHEC
Irish Centre for High-End Computing

Cancellation

- Cancellation of a construct. Threads/tasks canceled and execution continues after the end of the construct.

C/C++:
```
#pragma omp cancel [clauses]
```

Fortran:
```
!$omp cancel [clauses]
```

- Clauses: parallel, sections, for/do, taskgroup.
- Cancellation point: Allow users to explicitly define a cancellation point at which a check is done if cancellation has been requested/enabled, then, cancellation is performed.

C/C++:
```
#pragma omp cancelltionpoint
[clause]
```

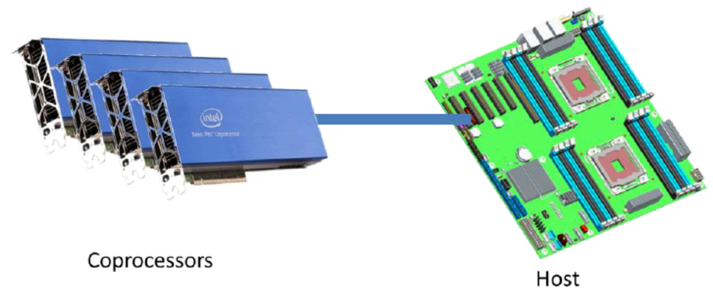
Fortran:
```
!$omp cancellationpoint [clause]
```

- Cancellation is performed if OMP_CANCELLATION is set to true

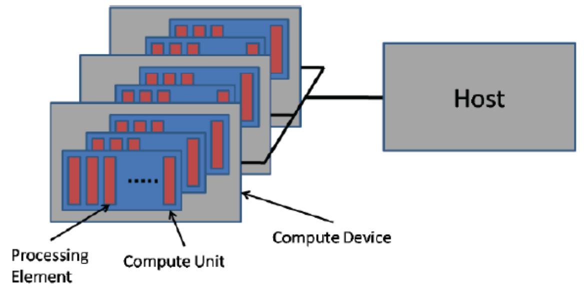
ICHEC Irish Centre for High-End Computing

Accelerators Support …(1)

- Enables the usage of accelerators and coprocessors to offload computation Ex: Intel Xeon Phi, NVIDIA GPUs
- Host device: The device on which the OpenMP program begins execution.
- Target device: A device onto which code and data may be offloaded from the host device.
- The execution of an OpenMP program starts on the host device and it may offload target regions to target device.



Host and Co-processors



Host and GPUs

ICHEC Irish Centre for High-End Computing

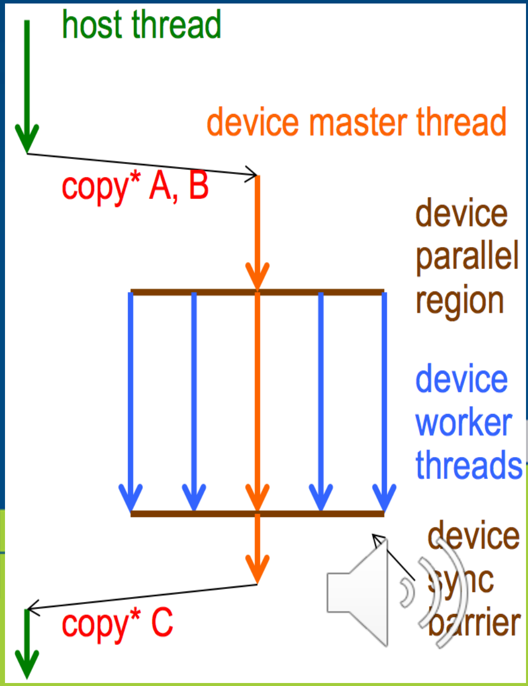
Accelerators Support ...(2)

- Marks a region to execute on target device

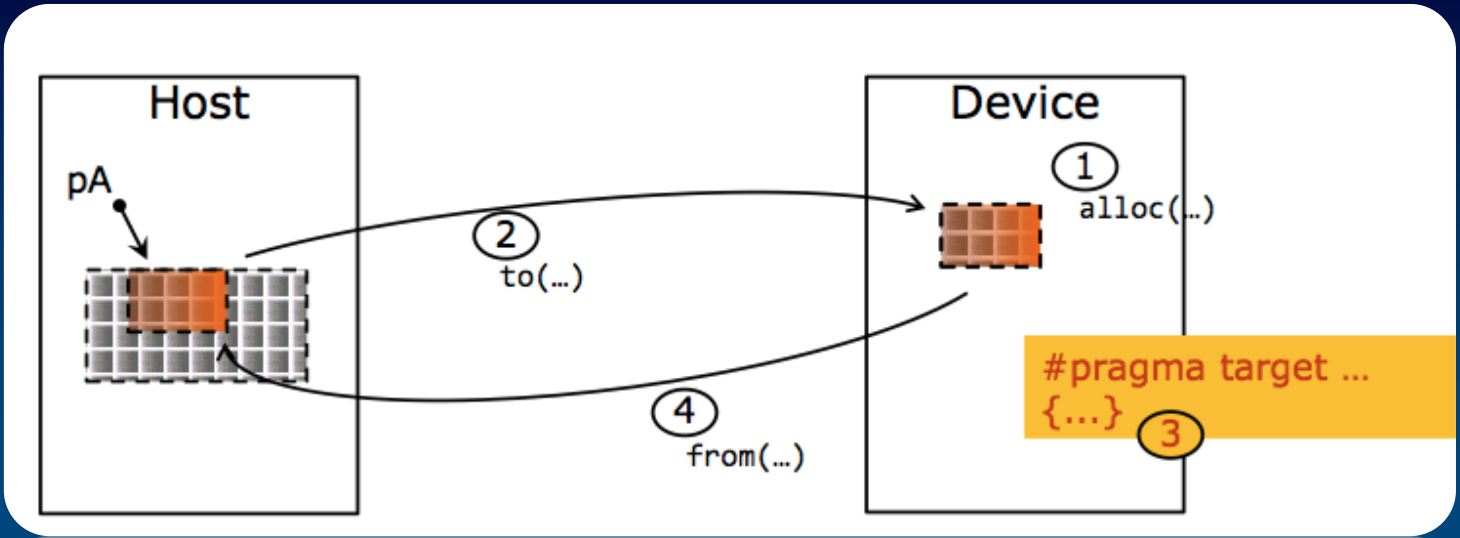
C/C++:
```
#pragma omp target [clauses]
 …
```

Fortran:
```
!$omp target [clauses]
   …
!$omp end target
```

- Target region is executed on one target device; each device has its own threads.
- Host thread waits until offloaded region completes.
- declare target construct: For functions/subroutines
- Clauses: if, nowait, device(expression), map (*[map-type:] list*)
  - if: if present and false, the device is the host
  - nowait: remove the implicit barrier
  - device: specify the type of device
  - map: maps a variable from the current task's data environment to the device data environment



Accelerators Support ...(3)



- alloc: each new corresponding list item has an undefined initial value
- to: each new corresponding list item is initialized with the original list item's value
- from: on exit from the region the corresponding list item's value is assigned to the original list item
- tofrom: both from and to (default)

OpenMP 4.5

OpenMP 4.5 released in November 2015

- Significantly improved support for accelerators
- Extensions to tasking, reductions, thread affinity, simd, loop
- Additional clauses
- Improved Support for Fortran 2003.

ICHEC Irish Centre for High-End Computing

OpenMP 5.0

OpenMP 5.0 released in November 2018

- Full support for accelerators
- Improved debugging and performance analysis
- Support for important features of the latest versions of C/C++ and Fortran.
- Support for a fully descriptive loop construct

Plans for further releases: v5.x on Nov 2020, v6.0 on Nov 2023.

https://www.openmp.org/