Advanced OpenMP
Title
archer2
epcc

OpenMP 4.0 and later

- Version 4.0 was released in July 2013
  - accelerator offloading, thread affinity, more task support
- Version 4.5 was released in Nov 2015
  - corrections and a few new features
  - now available in most production compilers (apart from the accelerator offloading features)
- Version 5.0 was released in Nov 2018
  - some extra features, extensions to offloading, new memory model, recent base language version support
  - some implementations support this
- Version 5.1 was released in Nov 2020
  - even more features
  - implementation still mostly work in progress

Overview

- User defined reductions
- Construct cancellation
- Portable SIMD directives
- Extensions to tasking
- Thread affinity

User defined reductions

- As of 3.1 cannot do reductions on objects or structures.
- UDR extensions in 4.0 added support for this.

- Use **`declare reduction`** directive to define new reduction operators
- New operators can then be used in reduction clause.

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

- `reduction-identifier` gives a name to the operator
  - Can be overloaded for different types
  - Can be redefined in inner scopes
- `typename-list` is a list of types to which it applies
- `combiner` expression specifies how to combine values
- `identity` can specify the identity value of the operator
  - Can be an expression or a brace initializer

Example: merging vectors

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

- Private copies created for a reduction are initialized to the identity that was specified for the operator and type
  - Default identity defined if identity clause not present
- Compiler uses combiner to combine private copies
- **`omp_out`** refers to private copy that holds combined values
- **`omp_in`** refers to the other private copy
- Can now use **`merge`** as a reduction operator.

Construct cancellation

- Clean way to signal early termination of an OpenMP construct.
  - one thread signals
  - other threads jump to the end of the construct

```
!$omp cancel construct [if (expr)]
```

where *construct* is **`parallel`**, **`sections`**, **`do`** or **`taskgroup`** cancels the construct

```
!$omp cancellation point construct
```

checks for cancellation (also happens implicitly at cancel directive, barriers etc.)

Example

```
!$omp parallel do private(eureka)
do i=1,n
   eureka = testing(i,...)
!$omp cancel parallel if(eureka)
end do
```

- First thread for which `eureka` is true will cancel the parallel region and exit.
- Other threads exit next time they hit the `cancel` directive

Portable SIMD directives

- Many compilers support SIMD directives to aid vectorisation of loops.
  - compiler can struggle to generate SIMD code without these
- OpenMP 4.0 provides a standardised set
- Use **`simd`** directive to indicate a loop should be SIMDized

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

- Executes iterations of following loop in SIMD chunks
- Loop is not divided across threads
- SIMD chunk is set of iterations executed concurrently by SIMD lanes

- Clauses control data environment, how loop is partitioned
- `safelen(length)` limits the number of iterations in a SIMD chunk.
- `linear` lists variables with a linear relationship to the iteration space
- `aligned` specifies byte alignments of a list of variables
- `private`, `lastprivate`, `reduction` and `collapse` have usual meanings.
- Also `declare simd` directive to generate SIMDised versions of functions.
- Can be combined with loop constructs (parallelise and SIMDise)

Extensions to tasking

`taskloop` directive provides a convenient way to turn loop iterations into tasks

```
#pragma omp taskloop grainsize(16)
for (int i=0; i<n; i++){
   a[i] = somefunc(i);
}
```

- Each set of 16 consecutive iterations will form a task
- Can specify the number of tasks with the **`num_tasks`** clause instead of grainsize

Task dependencies

- **`depend`** clause on task construct

`!$omp task depend(`*type*`:`***list*`)`**

where *type* is **`in`**, **`out`** or **`inout`** and *list* is a list of variables.
  - list may contain subarrays: OpenMP 4.0 includes a syntax for C/C++
- **`in`**: the generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items in an **`out`** or **`inout`** clause.
- **`out`** or **`inout`**: the generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items in **`in`**, **`out`** or **`inout`** clause.

Example

```
#pragma omp task depend (out:a)
    { ... }
#pragma omp task depend (out:b)
    { ... }
#pragma omp task depend (in:a,b)
    { ... }
```

- The first two tasks can execute in parallel
- The third task cannot start until both the first two are complete

Thread affinity

- OpenMP 4.0 gives much more control

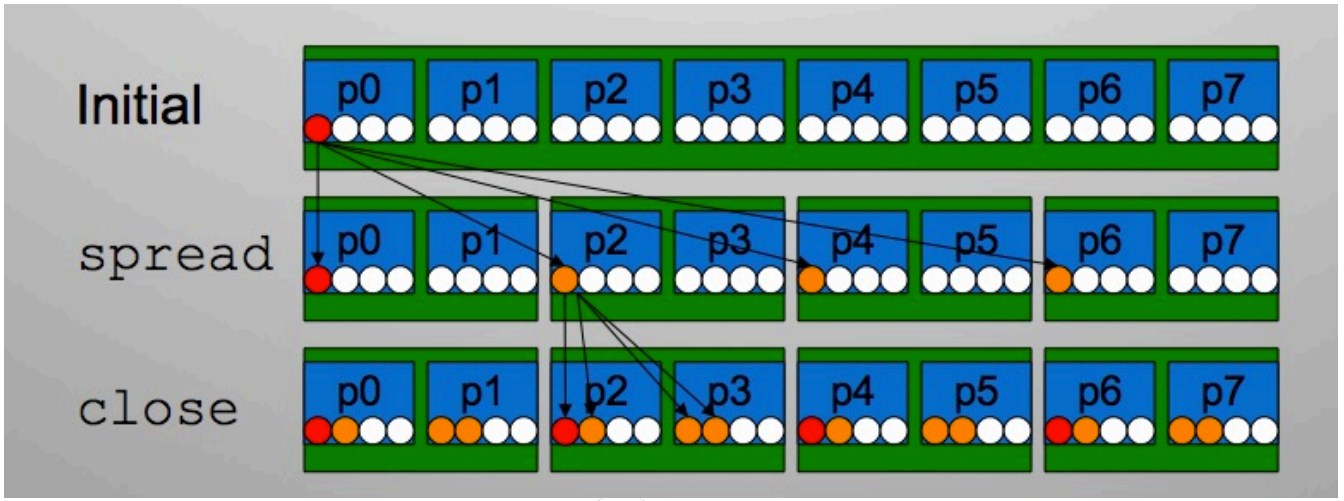
Affinity environment

- Since many systems are now NUMA and SMT, placement of threads on the hardware can have a big effect on performance.
- Increased choices for `OMP_PROC_BIND`
- Can still specify `true` or `false`
- Can now provide a list (possible item values: `master`, `close` or `spread`) to specify how to bind parallel regions at different nesting levels.
- Added `OMP_PLACES` environment variable
- Can specify abstract names including threads, cores and sockets
- Can specify an explicit ordered list of places
- Place numbering is implementation defined

Example

- Two levels of nested parallel regions
- Hardware with 8 cores and 4 hardware threads per core

```
export OMP_NUM_THREADS=4,2
export OMP_PLACES=threads
export OMP_PROC_BIND="spread,close"
```



Reusing this material

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

https://creativecommons.org/licenses/by-nc-sa/4.0/

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.