Introduction to OpenMP

OpenMP in small bites: Loops with Tasks

Dr. Christian Terboven

i12 High Performance Computing
RWTH AACHEN UNIVERSITY

Loops with Tasks

The taskloop Construct

- Task generating construct: decompose a loop into chunks, create a task for each loop chunk

```
#pragma omp taskloop [clause[[,] clause]…]
{structured-for-loops}
```

  - Where clause is one of:

Data Environment
- shared(list)
- private(list)
- firstprivate(list)
- lastprivate(list)
- default(sh | *pr* | *fp* | none)
- reduction(r-id: list)
- in_reduction(r-id: list)

Chunks/Grain
- grainsize(grain-size)
- num_tasks(num-tasks)

Cutoff Strategies
- if(scalar-expression)
- final(scalar-expression)
- mergeable

Scheduler (R/H)
- untied
- priority(priority-value)

Miscellaneous
- collapse(n)
- nogroup
- allocate([allocator:] list)

Worksharing vs. taskloop constructs (1/2)

```
subroutine worksharing
    integer :: x
    integer :: i
    integer, parameter :: T = 16
    integer, parameter :: N = 1024

    x = 0
!$omp parallel shared(x) num_threads(T)

!$omp do
    do i = 1,N
!$omp atomic
        x = x + 1
!$omp end atomic
    end do
!$omp end do

!$omp end parallel
    write (*,'(A,I0)') 'x = ', x
end subroutine
```

Result: x = 1024

```
subroutine taskloop
    integer :: x
    integer :: i
    integer, parameter :: T = 16
    integer, parameter :: N = 1024

    x = 0
!$omp parallel shared(x) num_threads(T)

!$omp taskloop
    do i = 1,N
!$omp atomic
        x = x + 1
!$omp end atomic
    end do
!$omp end taskloop

!$omp end parallel
    write (*,'(A,I0)') 'x = ', x
end subroutine
```

Result: x = 16384

Worksharing vs. taskloop constructs (2/2)

```
subroutine worksharing
    integer :: x
    integer :: i
    integer, parameter :: T = 16
    integer, parameter :: N = 1024

    x = 0
!$omp parallel shared(x) num_threads(T)

!$omp do
    do i = 1,N
!$omp atomic
        x = x + 1
!$omp end atomic
    end do
!$omp end do

!$omp end parallel
    write (*,'(A,I0)') 'x = ', x
end subroutine
```

Result: x = 1024

```
subroutine taskloop
    integer :: x
    integer :: i
    integer, parameter :: T = 16
    integer, parameter :: N = 1024

    x = 0
!$omp parallel shared(x) num_threads(T)
!$omp single
!$omp taskloop
    do i = 1,N
!$omp atomic
        x = x + 1
!$omp end atomic
    end do
!$omp end taskloop
!$omp end single
!$omp end parallel
    write (*,'(A,I0)') 'x = ', x
end subroutine
```

Result: x = 1024

Taskloop decomposition approaches

- Clause: grainsize(grain-size)
  - → Chunks have at least grain-size iterations
  - → Chunks have maximum 2x grain-size iterations

```
int TS = 4 * 1024;
#pragma omp taskloop grainsize(TS)
for ( i = 0; i<SIZE; i+=1) {
   A[i]=A[i]*B[i]*S;
}
```

- Clause: num_tasks(num-tasks)
  - → Create num-tasks chunks
  - → Each chunk must have at least one iteration

```
int NT = 4 * omp_get_num_threads();
#pragma omp taskloop num_tasks(NT)
for ( i = 0; i<SIZE; i+=1) {
   A[i]=A[i]*B[i]*S;
}
```

- If none of previous clauses is present, the *number of chunks* and the *number of iterations per chunk* is implementation defined
- Additional considerations:
  - → The order of the creation of the loop tasks is unspecified
  - → Taskloop creates an implicit taskgroup region; **nogroup** → no implicit taskgroup region is created

Questions?