Programming OpenMP

Tasking: taskloop

Christian Terboven

Jannis Klinkenberg

Programming in OpenMP
Christian Terboven & Members of the OpenMP Language Committee

The `taskloop` Construct

Tasking use case: saxpy (taskloop)

```c
for ( i = 0; i<SIZE; i+=1) {
   A[i]=A[i]*B[i]*S;
}
```

```c
for ( i = 0; i<SIZE; i+=TS) {
   UB = SIZE < (i+TS)?SIZE:i+TS;
   for ( ii=i; ii<UB; ii++) {
      A[ii]=A[ii]*B[ii]*S;
   }
}
```

```c
#pragma omp parallel
#pragma omp single
for ( i = 0; i<SIZE; i+=TS) {
   UB = SIZE < (i+TS)?SIZE:i+TS;
   #pragma omp task private(ii) \
    firstprivate(i,UB) shared(S,A,B)
   for ( ii=i; ii<UB; ii++) {
      A[ii]=A[ii]*B[ii]*S;
   }
}
```

- Difficult to determine grain
  - → 1 single iteration → to fine
  - → whole loop → no parallelism
- Manually transform the code
  - → blocking techniques
- Improving programmability
  - → OpenMP taskloop

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

- → Hiding the internal details
- → Grain size ~ Tile size (TS) → but implementation decides exact grain size

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}
```

```
!$omp taskloop [clause[[,] clause]…]
…structured-do-loops…
!$omp end taskloop
```

- Where clause is one of:

→ shared(list)

→ private(list)

→ firstprivate(list)

→ lastprivate(list)

→ default(sh | *pr* | *fp* | none)

→ reduction(r-id: list)

→ in_reduction(r-id: list)

Data Environment

→ grainsize(*[strict:]*grain-size)

→ num_tasks(*[strict:]*num-tasks)

Chunks/Grain

→ if(scalar-expression)

→ final(scalar-expression)

→ mergeable

Cutoff Strategies

→ untied

→ priority(priority-value)

Scheduler (R/H)

→ collapse(n)

→ nogroup

→ allocate([allocator:] list)

Miscellaneous

Worksharing vs. taskloop constructs (1/2)

```fortran
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

```fortran
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

Programming in OpenMP
Christian Terboven & Members of the OpenMP Language Committee

Worksharing vs. taskloop constructs (2/2)

```fortran
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

```fortran
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

■ `grainsize([strict:]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;
}
```

■ `num_tasks([strict:]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

Programming in OpenMP
Christian Terboven & Members of the OpenMP Language Committee

Collapsing iteration spaces with taskloop

■ The collapse clause in the taskloop construct

```
#pragma omp taskloop collapse(n)
{structured-for-loops}
```

→ Number of loops associated with the taskloop construct (n)

→ Loops are collapsed into one larger iteration space

→ Then divided according to the **grainsize** and **num_tasks**

■ Intervening code between any two associated loops

→ at least once per iteration of the enclosing loop

→ at most once per iteration of the innermost loop

```
#pragma omp taskloop collapse(2)
for ( i = 0; i<SX; i+=1) {
    for (  j= 0; j<SY; j+=1) {
        for ( k = 0; k<SZ; k+=1) {
            A[f(i,j,k)]=<expression>;
        }
    }
}
```

```
#pragma omp taskloop
for ( ij = 0; ij<SX*SY; ij+=1) {
    for ( k = 0; k<SZ; k+=1) {
        i = index_for_i(ij);
        j = index_for_j(ij);
        A[f(i,j,k)]=<expression>;
    }
}
```

Task reductions (using taskloop)

- Clause: `reduction(r-id: list)`
  - → It defines the scope of a new reduction
  - → All created tasks participate in the reduction
  - → It cannot be used with the **nogroup** clause

```c
double dotprod(int n, double *x, double *y) {
    double r = 0.0;
    #pragma omp taskloop in_reduction(+: r)
    for (i = 0; i < n; i++)
        r += x[i] * y[i];

    return r;
}
```

- Clause: `in_reduction(r-id: list)`
  - → Reuse an already defined reduction scope
  - → All created tasks participate in the reduction
  - → It can be used with the **nogroup*** clause, but it
    is user responsibility to guarantee result

```c
double dotprod(int n, double *x, double *y) {
    double r = 0.0;
    #pragma omp taskgroup task_reduction(+: r)
    {
        #pragma omp taskloop in_reduction(+: r)*
        for (i = 0; i < n; i++)
            r += x[i] * y[i];
    }
    return r;
}
```

Composite construct: taskloop simd

- Task generating construct: decompose a loop into chunks, create a task for each loop chunk
- Each generated task will apply (internally) SIMD to each loop chunk

  → C/C++ syntax:

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

  → Fortran syntax:

  ```
  !$omp taskloop simd [clause[[,] clause]…]
  …structured-do-loops…
  !$omp end taskloop
  ```

- Where clause is any of the clauses accepted by **taskloop** or **simd** directives