OpenMP Compiler Support

Marc Gonzàlez Tallada

Dept. d’Arquitectura de Computadors

Universitat Politècnica de Catalunya

OpenMP Compiler Support

- **Parallelism definition**
  - OpenMP follows an SPMD execution model
    - ✓ All threads execute the same code
    - ✓ Parallel code has to be transformed
      - – Work distribution
      - – Variable scoping
      - – Synchronizations

- **Compiler Transformations**

?

OpenMP Compiler Support

Parallelism definition

- OpenMP follows an SPMD execution model
  - ✓ All threads execute the same code
  - ✓ Parallel code has to be transformed
    - – Work distribution
    - – Variable scoping
- Compiler Transformations
  - ✓ Parallel code is encapsulated in a function

OpenMP Compiler Support

Parallelism definition

- OpenMP follows an SPMD execution model
  - ✓ All threads execute the same code
  - ✓ Parallel code has to be transformed
    - – Work distribution
    - – Variable scoping
    - – Thread synchronizations

- Compiler Transformations
  - ✓ Parallel code is encapsulated in a function
  - ✓ Parallel code is modified with
    - – Runtime calls
    - – Add/remove thread symbols

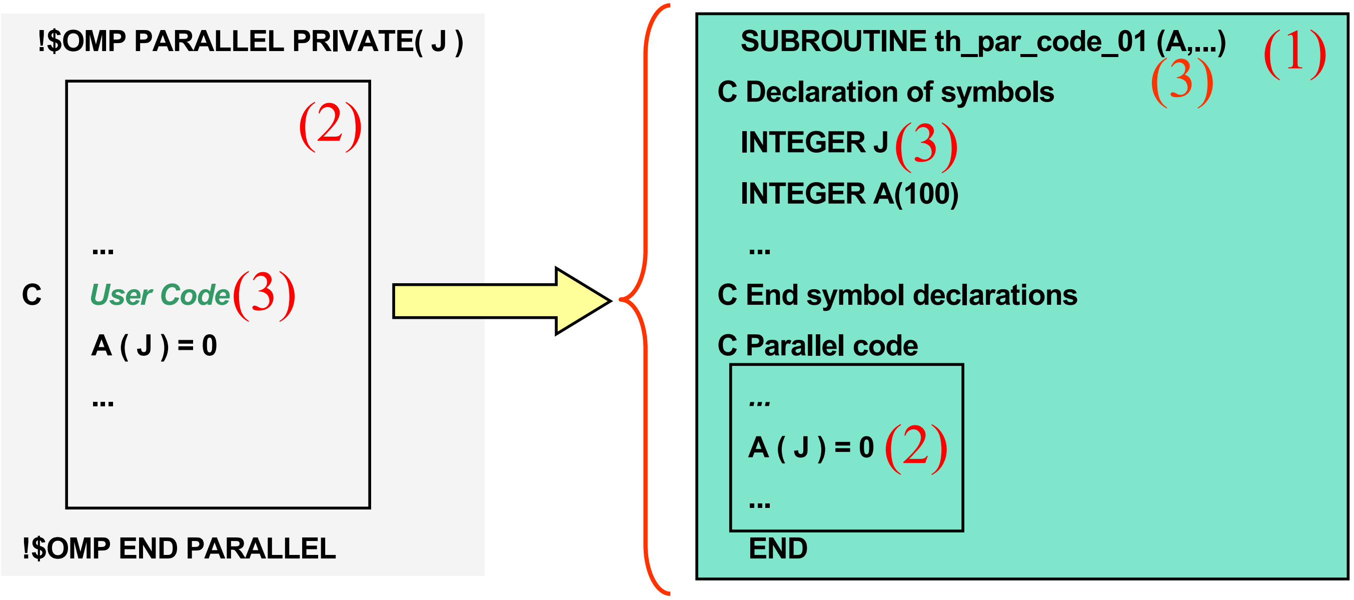
Generic Runtime Support

- **Parallelism definition**
  - rtl_get_num_threads ( )
  - rtl_create_thread ()
  - rtl_join()
- **Synchronization**
  - rtl_spin_lock ()
  - rtl_spin_unlock ()
  - rtl_barrier ()
  - Atomic operations
    - ✓ rtl_atm_add_4 ()
    - ✓ rtl_atm_add_8 ()
    - ✓ ...

OpenMP Compiler Support

Parallelism definition

- Parallel code encapsulation
  - (1) Definition of the thread function
    - (2) Extract parallel code
      - (3) Gather symbols
      - (3) Filter symbols: PRIVATE, Global variables

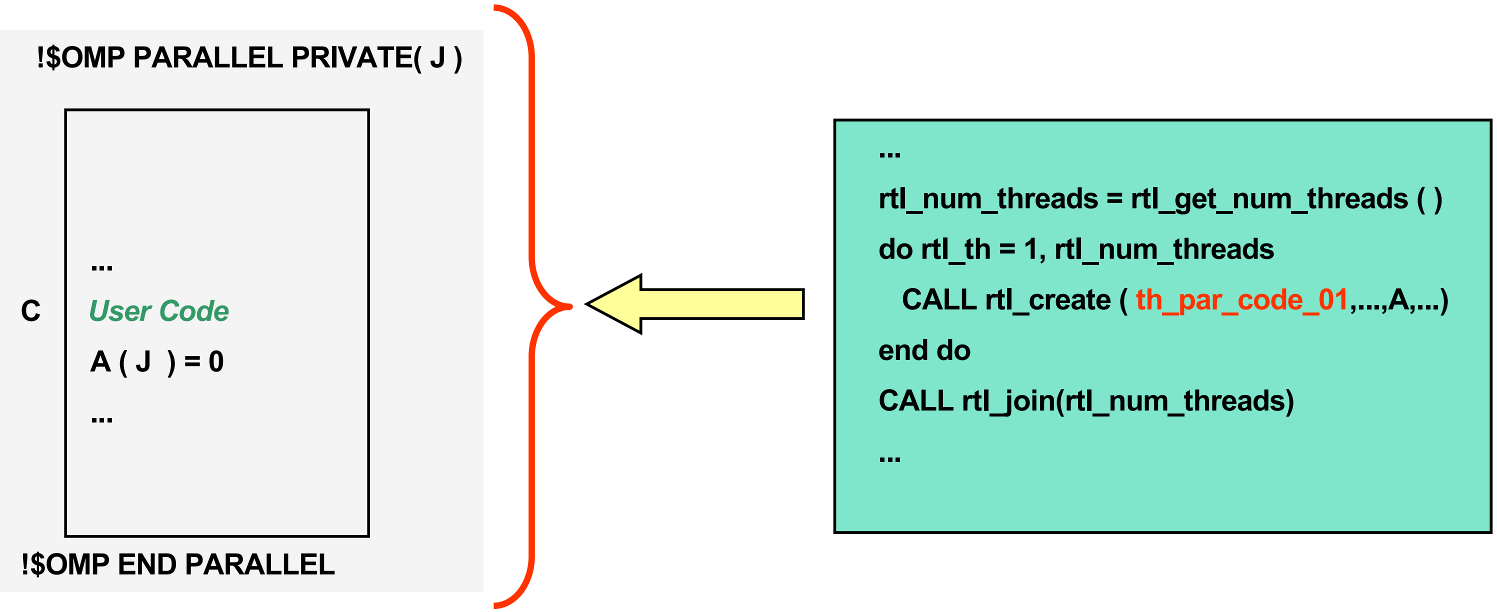


```
!$OMP PARALLEL PRIVATE( J )
    ...
C   User Code
    A ( J ) = 0
    ...
!$OMP END PARALLEL
```

```
  SUBROUTINE th_par_code_01 (A,...)
C Declaration of symbols
  INTEGER J
  INTEGER A(100)
  ...
C End symbol declarations
C Parallel code
   ...
   A ( J ) = 0
   ...
  END
```

OpenMP Compiler Support

- **Parallelism definition**
  - Thread creation
    - ✓ Inject runtime calls
      - – Substitute the parallel region by the thread creation code

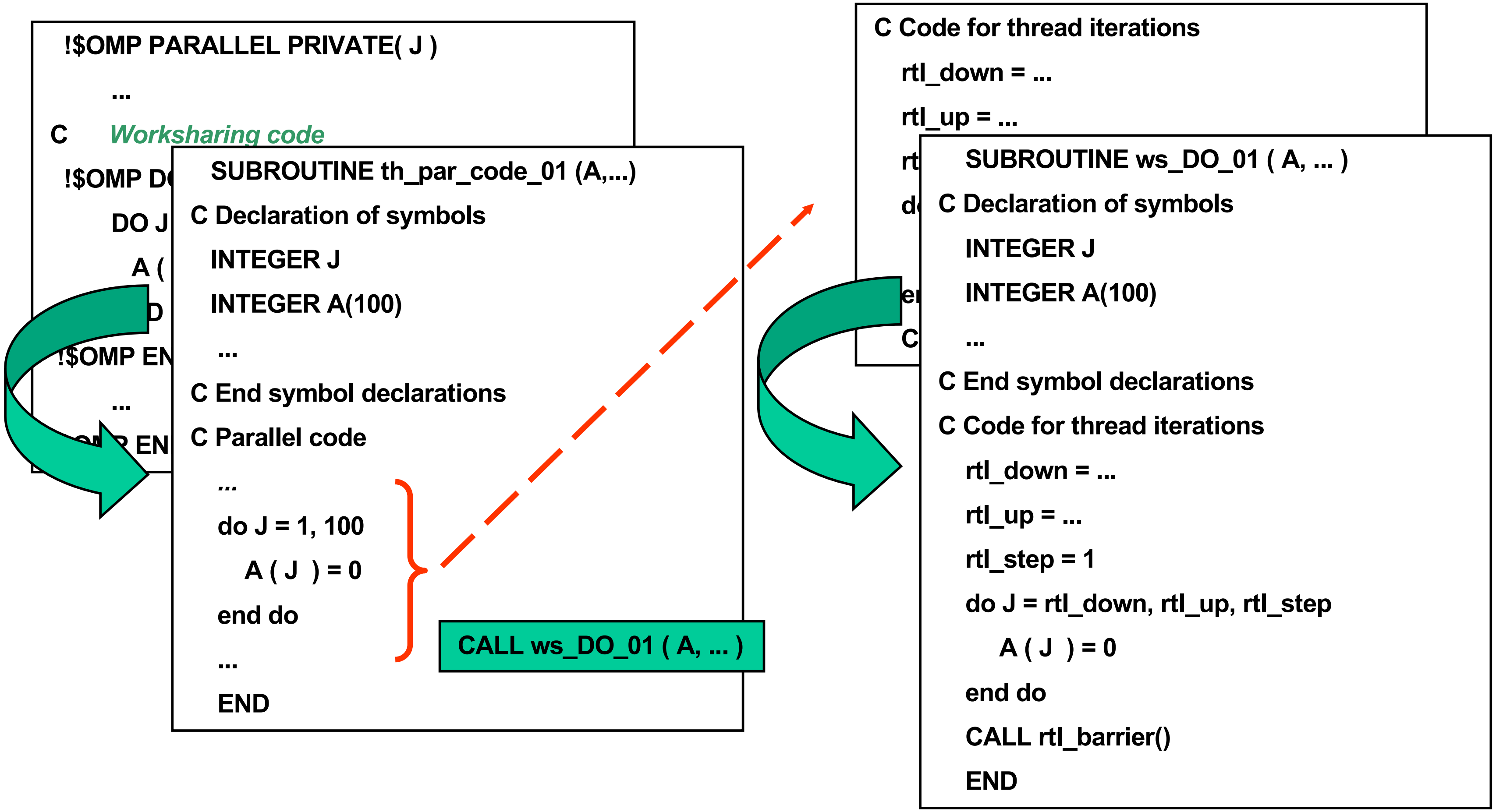


```
 !$OMP PARALLEL PRIVATE( J )

     ...
C    User Code
     A ( J ) = 0
     ...

!$OMP END PARALLEL
```

```
...
rtl_num_threads = rtl_get_num_threads ( )
do rtl_th = 1, rtl_num_threads
  CALL rtl_create ( th_par_code_01,...,A,...)
end do
CALL rtl_join(rtl_num_threads)
...
```

OpenMP Compiler Support

- **Work distribution**
  - DO worksharing construct



```
 !$OMP PARALLEL PRIVATE( J )
      ...
C     Worksharing code
 !$OMP D
      DO J
        A (
          D
 !$OMP EN
      ...
      EN
```

```
  SUBROUTINE th_par_code_01 (A,...)
C Declaration of symbols
  INTEGER J
  INTEGER A(100)
  ...
C End symbol declarations
C Parallel code
   ...
   do J = 1, 100
      A ( J  ) = 0
   end do
   ...
   END
```

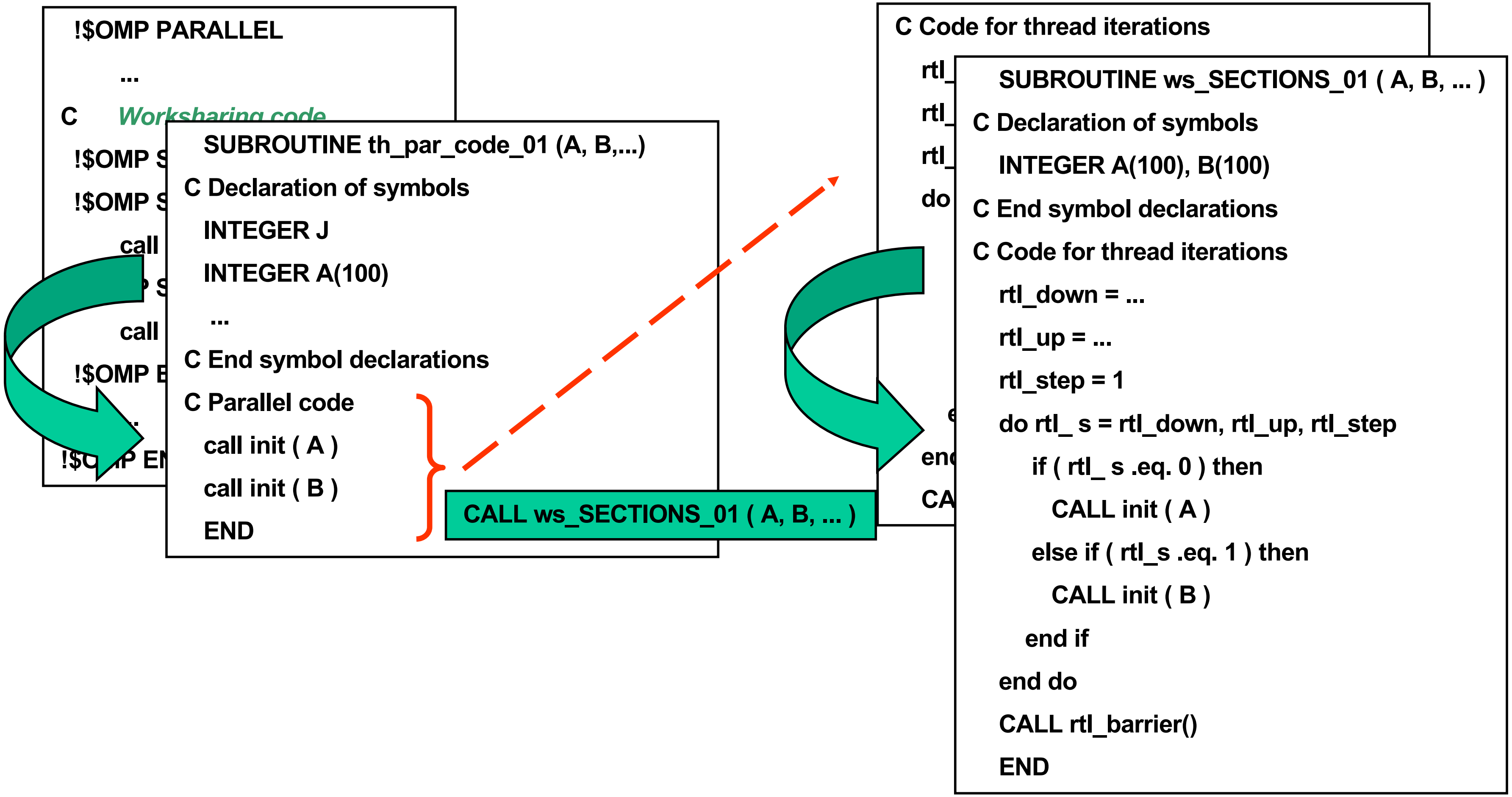
```
CALL ws_DO_01 ( A, ... )
```

```
C Code for thread iterations
  rtl_down = ...
  rtl_up = ...
  rt
  d
  e
  C
```

```
  SUBROUTINE ws_DO_01 ( A, ... )
C Declaration of symbols
  INTEGER J
  INTEGER A(100)
  ...
C End symbol declarations
C Code for thread iterations
  rtl_down = ...
  rtl_up = ...
  rtl_step = 1
  do J = rtl_down, rtl_up, rtl_step
     A ( J  ) = 0
  end do
  CALL rtl_barrier()
  END
```

OpenMP Compiler Support

- **Work distribution**
  - SECTIONS worksharing construct



```
  !$OMP PARALLEL
       ...
C      Worksharing code
  !$OMP S
  !$OMP S
       call
      P S
       call
  !$OMP E
       ..
  !$O  P EN
```

```
    SUBROUTINE th_par_code_01 (A, B,...)
  C Declaration of symbols
    INTEGER J
    INTEGER A(100)
     ...
  C End symbol declarations
  C Parallel code
    call init ( A )
    call init ( B )
    END
```

```
CALL ws_SECTIONS_01 ( A, B, ... )
```

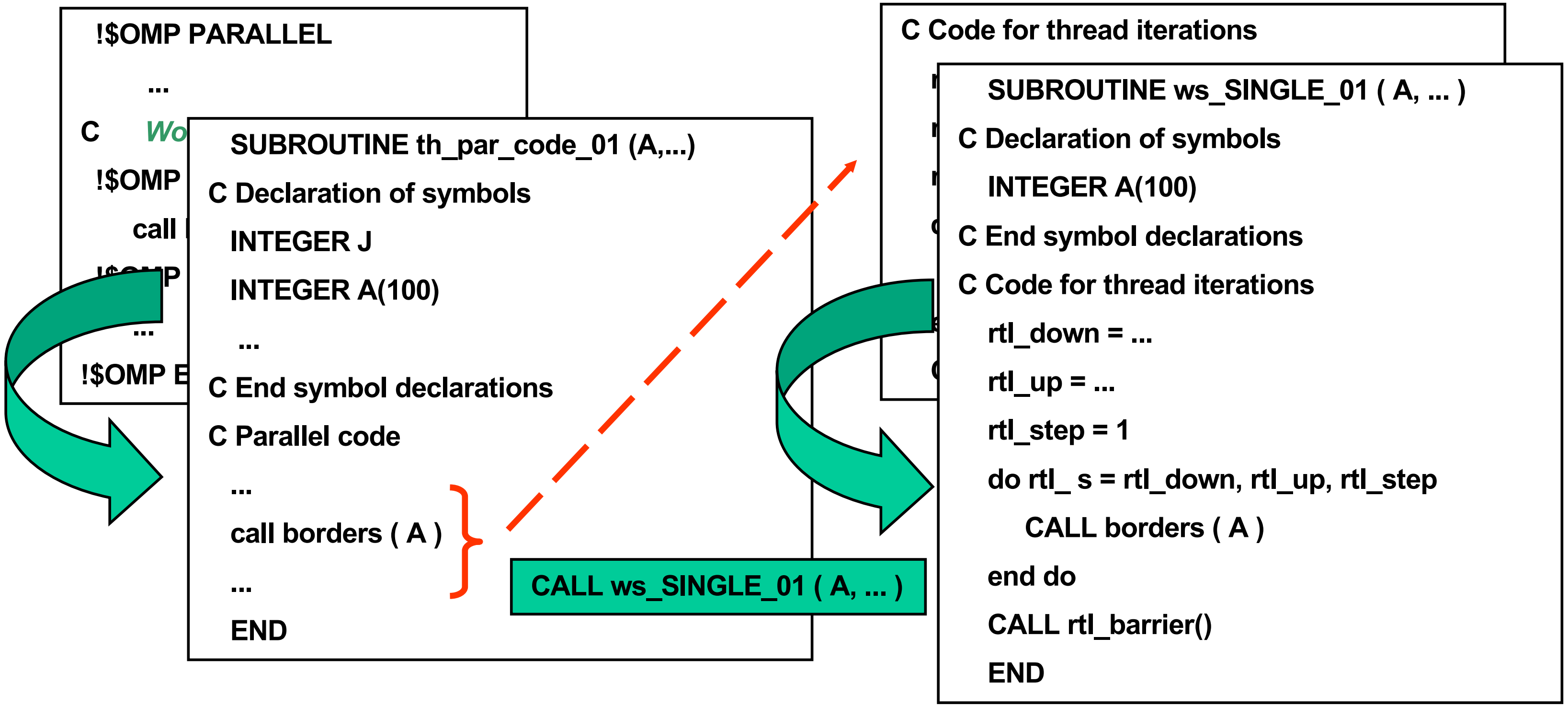
```
C Code for thread iterations
   rtl_
   rtl_
   rtl_
   do

       e
   end
   CA
```

```
    SUBROUTINE ws_SECTIONS_01 ( A, B, ... )
  C Declaration of symbols
    INTEGER A(100), B(100)
  C End symbol declarations
  C Code for thread iterations
    rtl_down = ...
    rtl_up = ...
    rtl_step = 1
    do rtl_ s = rtl_down, rtl_up, rtl_step
       if ( rtl_ s .eq. 0 ) then
          CALL init ( A )
       else if ( rtl_s .eq. 1 ) then
          CALL init ( B )
      end if
    end do
    CALL rtl_barrier()
    END
```

OpenMP Compiler Support

Work distribution

- SINGLE worksharing construct



```
!$OMP PARALLEL
      ...
C     Wo
!$OMP
   call
!$OMP
   ...
!$OMP E
```

```
  SUBROUTINE th_par_code_01 (A,...)
C Declaration of symbols
  INTEGER J
  INTEGER A(100)
   ...
C End symbol declarations
C Parallel code
  ...
  call borders ( A )
  ...
  END
```

```
CALL ws_SINGLE_01 ( A, ... )
```

```
C Code for thread iterations
```

```
  SUBROUTINE ws_SINGLE_01 ( A, ... )
C Declaration of symbols
  INTEGER A(100)
C End symbol declarations
C Code for thread iterations
  rtl_down = ...
  rtl_up = ...
  rtl_step = 1
  do rtl_ s = rtl_down, rtl_up, rtl_step
     CALL borders ( A )
  end do
  CALL rtl_barrier()
  END
```

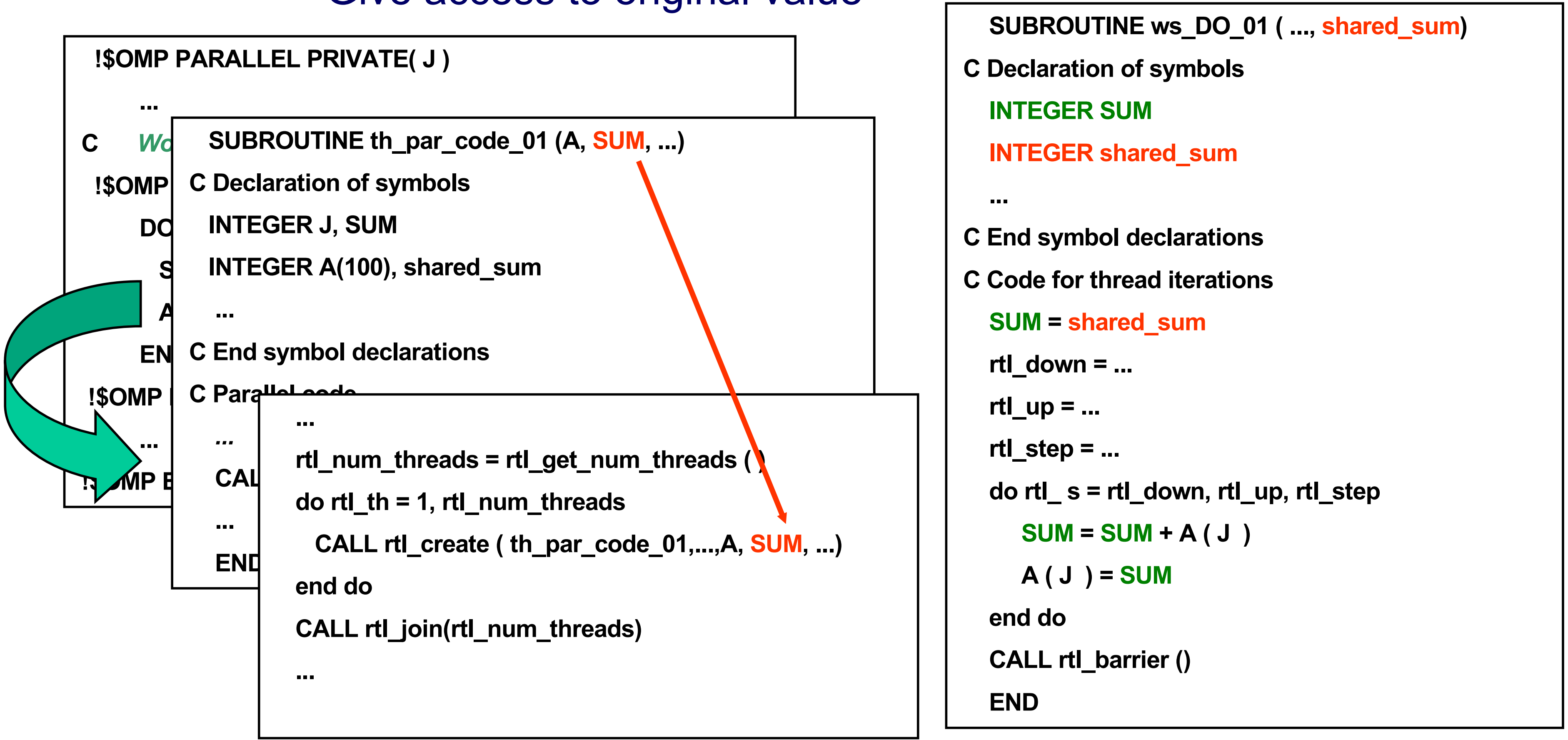
OpenMP Compiler Support

- **Variable scoping**
  - PRIVATE
    - ✓ Easy, in subroutine stack
  - SHARED
    - ✓ Default
    - ✓ Subroutine arguments

```
  SUBROUTINE th_par_code_01 (A,...)
C Declaration of symbols
  INTEGER J
  INTEGER A(100)
   ...
C End symbol declarations
C Parallel code
  ...
  A ( J  ) = ...
  ...
  END
```

OpenMP Compiler Support

- **Variable scoping**
  - FIRSTPRIVATE
    - ✓ Initialize private symbol
      - – Give access to original value



```
 !$OMP PARALLEL PRIVATE( J )
     ...
C    Wo
 !$OMP
     DO
       S
       A
     EN
 !$OMP
     ...
 !$OMP E
```

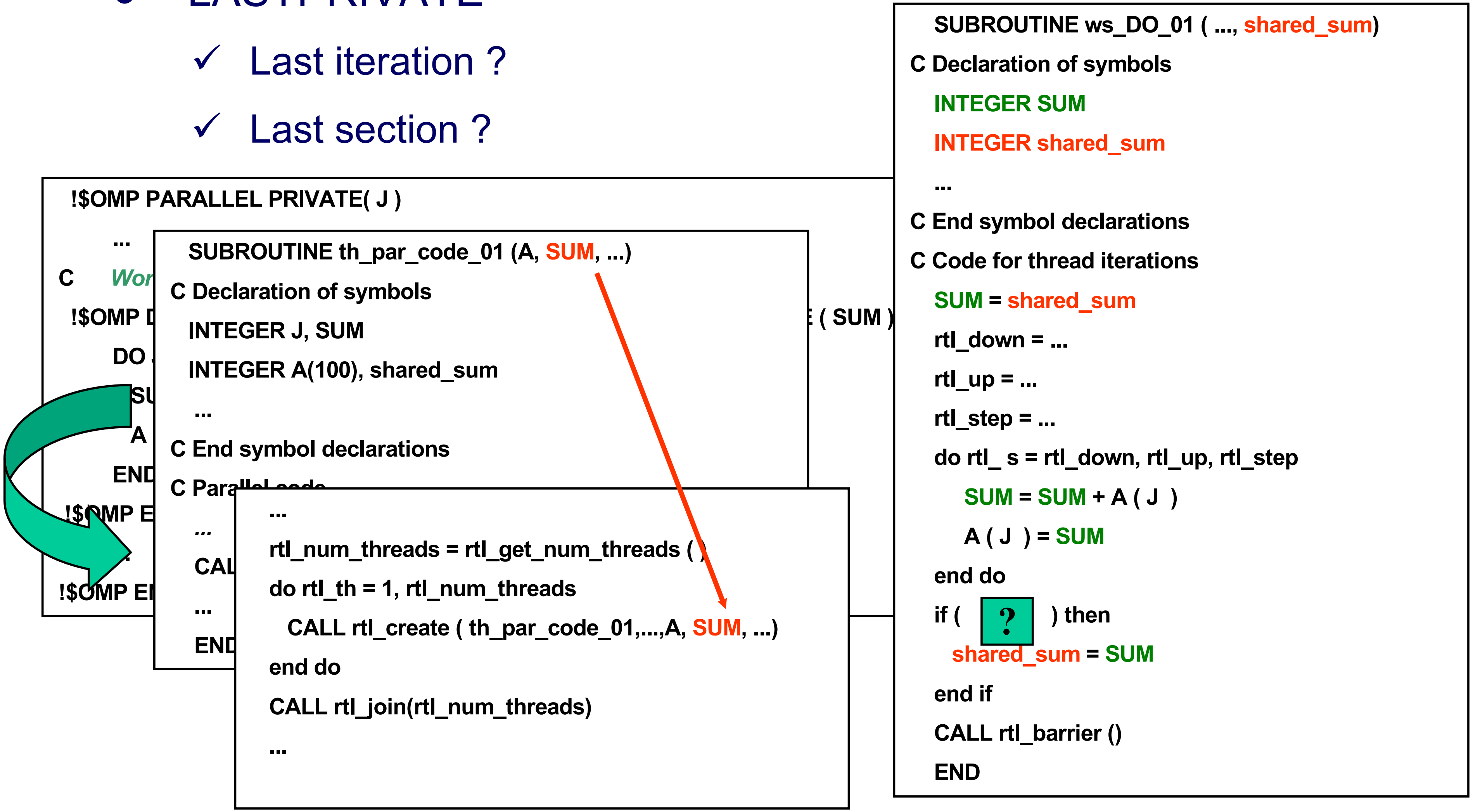
```
  SUBROUTINE th_par_code_01 (A, SUM, ...)
C Declaration of symbols
  INTEGER J, SUM
  INTEGER A(100), shared_sum
  ...
C End symbol declarations
C Parallel code
  ...
  CAL
  ...
  END
```

```
...
rtl_num_threads = rtl_get_num_threads ( )
do rtl_th = 1, rtl_num_threads
  CALL rtl_create ( th_par_code_01,...,A, SUM, ...)
end do
CALL rtl_join(rtl_num_threads)
...
```

```
  SUBROUTINE ws_DO_01 ( ..., shared_sum)
C Declaration of symbols
  INTEGER SUM
  INTEGER shared_sum
  ...
C End symbol declarations
C Code for thread iterations
  SUM = shared_sum
  rtl_down = ...
  rtl_up = ...
  rtl_step = ...
  do rtl_ s = rtl_down, rtl_up, rtl_step
     SUM = SUM + A ( J  )
     A ( J  ) = SUM
  end do
  CALL rtl_barrier ()
  END
```

OpenMP Compiler Support

- **Variable scoping**
  - LASTPRIVATE
    - ✓ Last iteration ?
    - ✓ Last section ?



```
 !$OMP PARALLEL PRIVATE( J )
    ...
C     Wor
 !$OMP D                                    ( SUM )
    DO
      SU
      A
    END
 !$OMP E
 !$OMP EI
```

```
   SUBROUTINE th_par_code_01 (A, SUM, ...)
C Declaration of symbols
   INTEGER J, SUM
   INTEGER A(100), shared_sum
   ...
C End symbol declarations
C Parallel code
   ...
   CAL
   ...
   END
```

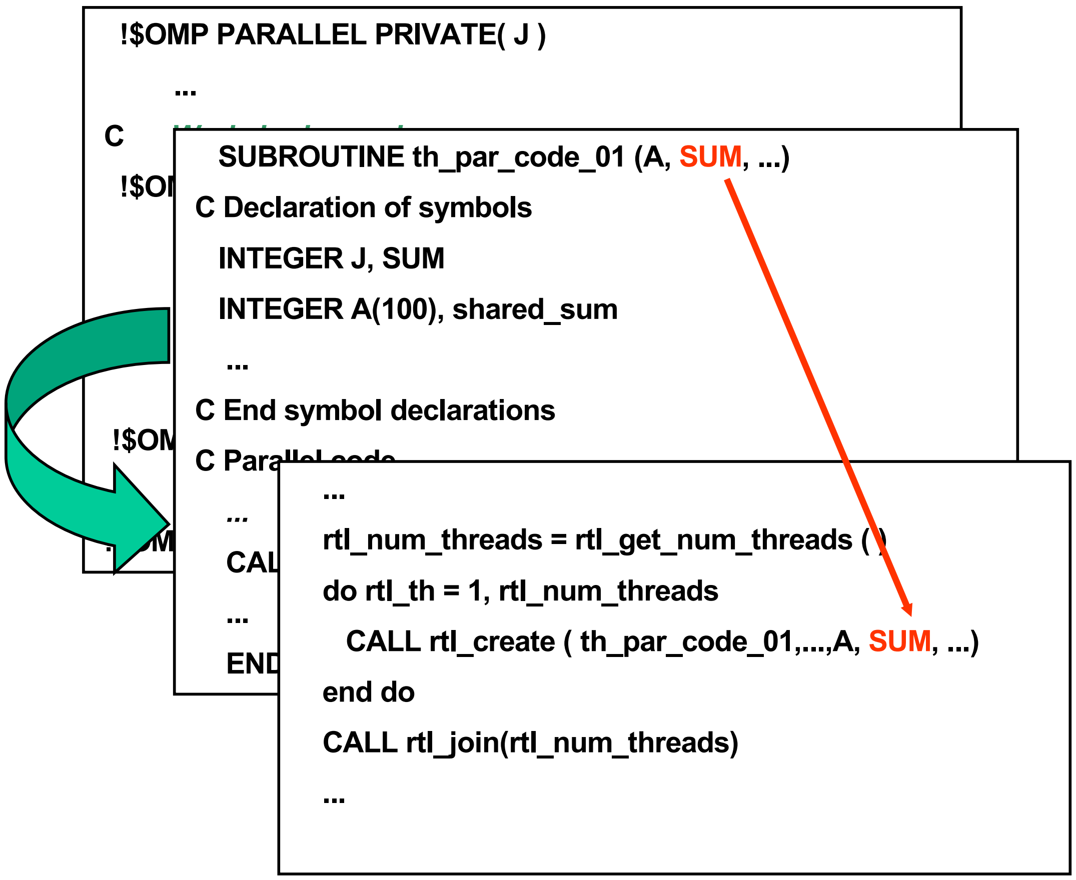
```
...
rtl_num_threads = rtl_get_num_threads ( )
do rtl_th = 1, rtl_num_threads
  CALL rtl_create ( th_par_code_01,...,A, SUM, ...)
end do
CALL rtl_join(rtl_num_threads)
...
```

```
  SUBROUTINE ws_DO_01 ( ..., shared_sum)
C Declaration of symbols
  INTEGER SUM
  INTEGER shared_sum
  ...
C End symbol declarations
C Code for thread iterations
  SUM = shared_sum
  rtl_down = ...
  rtl_up = ...
  rtl_step = ...
  do rtl_ s = rtl_down, rtl_up, rtl_step
     SUM = SUM + A ( J  )
     A ( J  ) = SUM
  end do
  if (  ?  ) then
    shared_sum = SUM
  end if
  CALL rtl_barrier ()
  END
```

OpenMP Compiler Support

Variable scoping

- REDUCTION



```
  !$OMP PARALLEL PRIVATE( J )
      ...
C
  !$O
 !$OM
 .OM
```

```
   SUBROUTINE th_par_code_01 (A, SUM, ...)
C Declaration of symbols
   INTEGER J, SUM
   INTEGER A(100), shared_sum
    ...
C End symbol declarations
C Parallel code
    ...
    CAL
    ...
    END
```

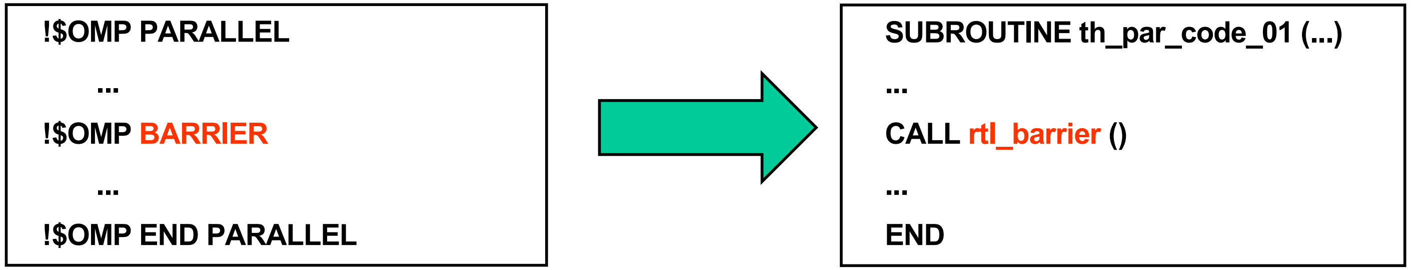
```
...
rtl_num_threads = rtl_get_num_threads ( )
do rtl_th = 1, rtl_num_threads
  CALL rtl_create ( th_par_code_01,...,A, SUM, ...)
end do
CALL rtl_join(rtl_num_threads)
...
```

```
   SUBROUTINE ws_do_01 ( A, shared_sum, ... )
C Declaration of symbols
   INTEGER J, SUM
   INTEGER A(100), shared_sum
   ...
C End symbol declarations
C Code for thread iterations
   SUM = "neuter of operation"
   rtl_down = ...
   rtl_up = ...
   rtl_step = 1
   do J = rtl_down, rtl_up, rtl_step
      SUM = SUM + A ( J  )
      A ( J  ) = 0
   end do
   CALL rtl_spin_lock ()
   shared_sum = shared_sum + SUM
   CALL rtl_spin_unlock ()
   CALL rtl_barrier()
   END
```

OpenMP Compiler Support

Synchronizations

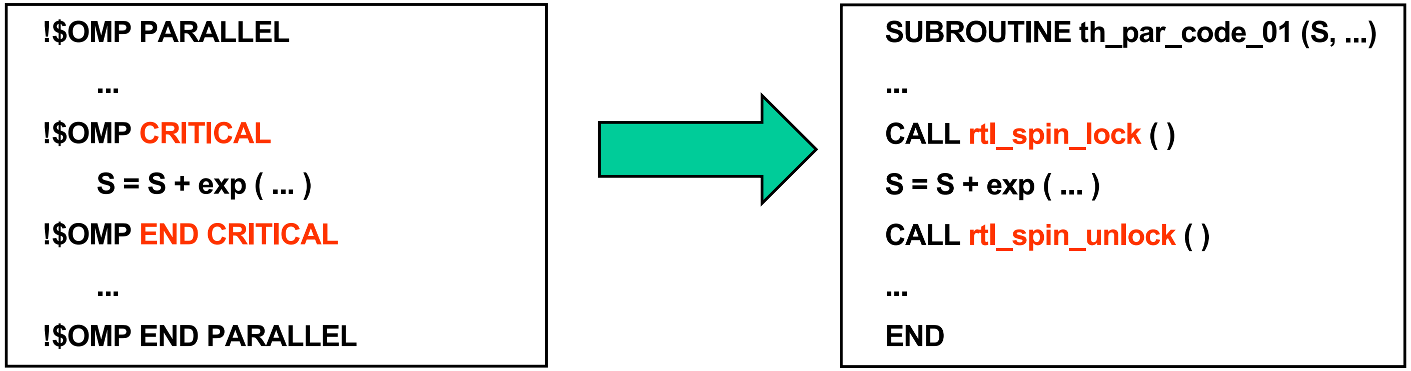
- BARRIER



```
!$OMP PARALLEL
      ...
!$OMP BARRIER
      ...
!$OMP END PARALLEL
```

```
SUBROUTINE th_par_code_01 (...)
...
CALL rtl_barrier ()
...
END
```

- CRITICAL

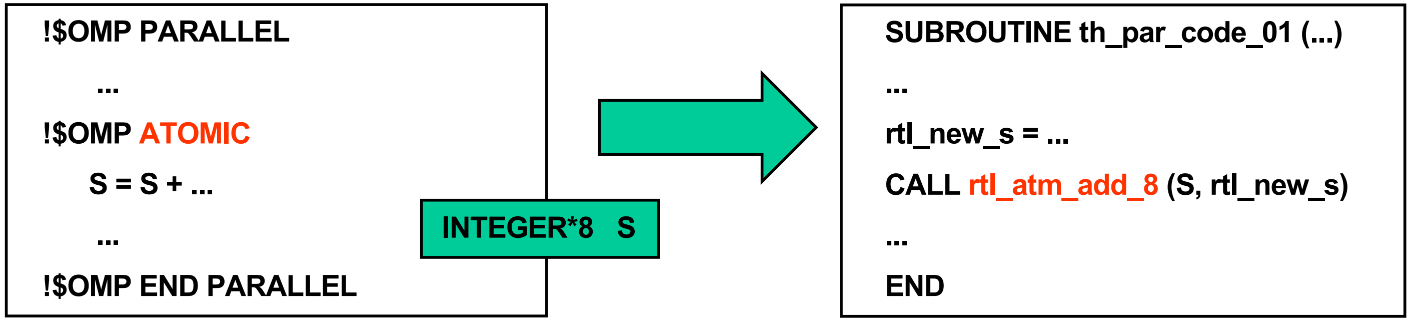


```
!$OMP PARALLEL
      ...
!$OMP CRITICAL
      S = S + exp ( ... )
!$OMP END CRITICAL
      ...
!$OMP END PARALLEL
```

```
SUBROUTINE th_par_code_01 (S, ...)
...
CALL rtl_spin_lock ( )
S = S + exp ( ... )
CALL rtl_spin_unlock ( )
...
END
```

OpenMP Compiler Support

- **Synchronizations**
  - ATOMIC
    - ✓ Size of the element
      - – 4, 8 bytes



```
!$OMP PARALLEL
      ...
!$OMP ATOMIC
      S = S + ...
      ...
!$OMP END PARALLEL
```

INTEGER*8 S

```
SUBROUTINE th_par_code_01 (...)
...
rtl_new_s = ...
CALL rtl_atm_add_8 (S, rtl_new_s)
...
END
```

Runtime Dependences

- **Parallelism definition**
  - Runtime support
    - ✓ ***nth_self***: returns a memory pointer to the thread descriptor of the invoking thread
    - ✓ ***nth_cpus_actual***: number of available threads
    - ✓ ***nth_depadd***: informs the runtime about howmany threads are going to execute the parallelism
    - ✓ ***nth_create_vp_1s***: creates a thread ready for execution
    - ✓ ***nth_block***: blocks the invoking thread until the parallelism termination
    - ✓ ***nth_whoami***: return the thread identifier in the team
    - ✓ ***nth_ami_master***: returns the appropriate boolean value, testing if the invoking thread is the master of the team

Runtime Dependences

Parallelism definition

```
!$OMP PARALLEL PRIVATE( J )

      ...
C     User Code
      A ( J ) = 0
      ...

!$OMP END PARALLEL
```

```
...
nth_self_p = nth_self ( )
nth_num_threads = nth_cpus_actual ( )
CALL nth_depadd ( nth_num_threads + 1 )
nth_mask = 608
do nth_p = 0,nth_num_threads-1
  nth = nth_create_vp_1s ( th_par_code_01, 0, nth_self_p, nth_p, nth_mask, 7, A, ...)
end do
CALL nth_block ( )
...
```

Obtains master thread descriptor

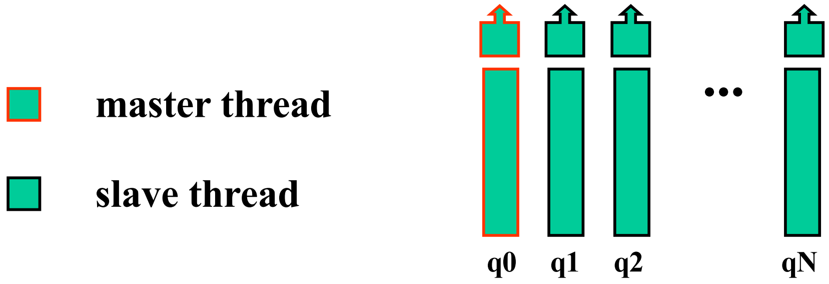
Number of threads

Bit mask for arguments: 01001001...

Number of arguments

Thread identifier

Master thread pointer



Runtime Dependences

- **Work distribution**
  - Worksharings:
    - ✓ All worksharings treated as parallel **do** loops
    - ✓ Basic runtime support:
      - ***begin_for***: informs the runtime about a loop definition

        first iteration

        last iteration

        loop step

        scheduling
      - ***end_for***: informs the runtime about a loop termination

        barrier flag
      - ***next_iters***: supplies the next piece of work

        next iterations to be executed

        last chunk of iterations

Runtime Dependences

- **Work distribution**
  - DO

```
 !$OMP PARALLEL PRIVATE( J )
      ...
C     Worksharing code
 !$OMP DO SCHEDULE( STATIC, 4 )
      DO J = 1, N
        A ( J ) = 0
      EN
 !$OMP E
      ...
!$OMP E
```

```
      SUBROUTINE th_par_code_01 (A,...)
C Declaration of symbols
      INTEGER J
      INTEGER A(100)
      ...
C End symbol declarations
C Parallel code
      ...
      CALL ws_DO_01 ( A, ... )
      ...
      END
```

```
   SUBROUTINE ws_DO_01 ( A, ... )
C Declaration of symbols
   INTEGER J
   INTEGER A(100)
   ...
C End symbol declarations
C Code for thread iterations
   CALL begin_for ( 1, N, 1, 01, 4 )
   do while ( next_iters ( nth_down, nth_up, nth_last ) .eqv. TRUE )
      do J = nth_down, nth_up, 1
         A ( J ) = 0
      end do
   end do
   CALL end_for ( 1 )
   END
```

Loop scheduling

Runtime Dependences

- **Work distribution**
  - SECTIONS

```
 !$OMP PARALLEL
     ...
C    Worksharing code
 !$OMP SECTIONS
 !$OMP SECTION
       call init ( A )
 !$OMP SECTION
       call init ( B )
 !$OMP E
     ...
 !$OMP E
```

```
   SUBROUTINE th_par_code_01 (A,B, ...)
C Declaration of symbols
   INTEGER J
   INTEGER A(100)
    ...
C End symbol declarations
C Parallel code
    ...
    CALL ws_SECTIONS_01 ( A, B, ... )
    ...
    END
```

```
   SUBROUTINE ws_SECTIONS_01 ( A, B, ... )
C Declaration of symbols
   INTEGER J
   INTEGER A(100)
   ...
C End symbol declarations
C Code for thread iterations
   CALL begin_for ( 0, 1, 1, 04, 1 )
   do while ( next_iters ( nth_down, nth_up, nth_last ) .eqv. TRUE )
       do nth_s = nth_down, nth_up, 1
         if ( nth_s .eq. 0 ) then
            CALL init ( A )
         else if ( nth_s .eq. 1 ) then
            CALL init ( B )
         end if
       end do
   end do
   CALL end_for ( 1 )
   END
```

DYNAMIC and CHUNK 1

Runtime Dependences

- **Work distribution**
  - SINGLE

```
 !$OMP PARALLEL
     ...
C    Worksharing code
 !$OMP SINGLE
      call borders ( A )
 !$OMP END SINGLE
     ..
 !$OM
```

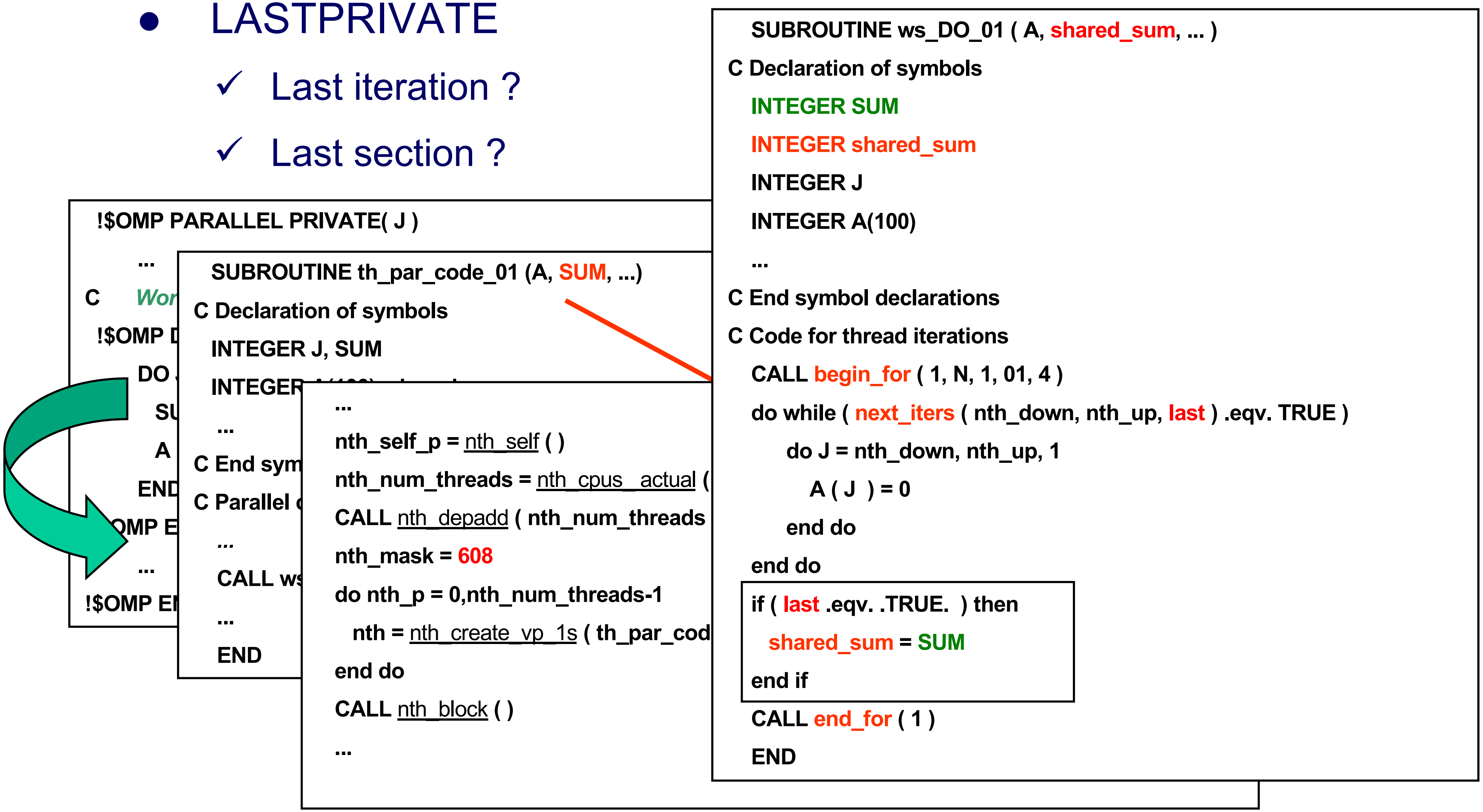
```
   SUBROUTINE th_par_code_01 (A,B, ...)
C Declaration of symbols
   INTEGER J
   INTEGER A(100)
    ...
C End symbol declarations
C Parallel code
    ...
    CALL ws_SINGLE_01 ( A, B, ... )
    ...
    END
```

```
   SUBROUTINE ws_SINGLE_01 ( A, B, ... )
C Declaration of symbols
   INTEGER J
   INTEGER A(100)
   ...
C End symbol declarations
C Code for thread iterations
   CALL begin_for ( 0, 0, 1, 04, 1 )
   do while ( next_iters ( nth_down, nth_up, nth_last ) .eqv. TRUE )
       do nth_s = nth_down, nth_up, 1
         if ( nth_s .eq. 0 ) then
             CALL borders ( A )
         end if
       end do
   end do
   CALL end_for ( 1 )
   END
```

DYNAMIC and CHUNK 1

Runtime Dependences

- **Variable scoping**
  - LASTPRIVATE
    - ✓ Last iteration ?
    - ✓ Last section ?



```
 !$OMP PARALLEL PRIVATE( J )
    ...
C     Wor
 !$OMP D
    DO
     SU
     A
    END
  OMP E
    ...
!$OMP EN
```

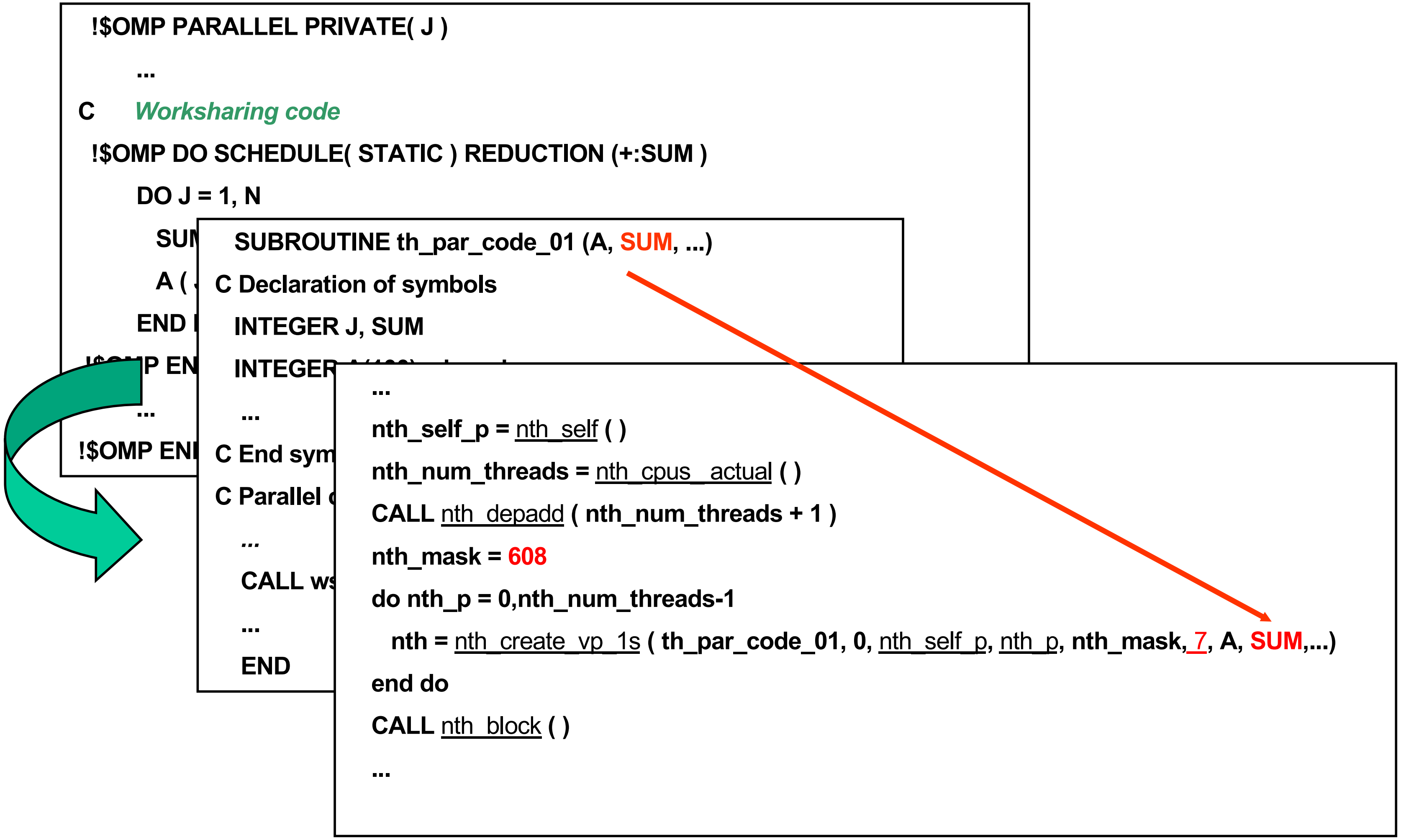
```
  SUBROUTINE th_par_code_01 (A, SUM, ...)
C Declaration of symbols
  INTEGER J, SUM
  INTEGER
   ...
C End sym
C Parallel c
   ...
   CALL ws
   ...
   END
```

```
  ...
  nth_self_p = nth_self ( )
  nth_num_threads = nth_cpus_actual (
  CALL nth_depadd ( nth_num_threads
  nth_mask = 608
  do nth_p = 0,nth_num_threads-1
    nth = nth_create_vp_1s ( th_par_cod
  end do
  CALL nth_block ( )
  ...
```

```
  SUBROUTINE ws_DO_01 ( A, shared_sum, ... )
C Declaration of symbols
  INTEGER SUM
  INTEGER shared_sum
  INTEGER J
  INTEGER A(100)
  ...
C End symbol declarations
C Code for thread iterations
  CALL begin_for ( 1, N, 1, 01, 4 )
  do while ( next_iters ( nth_down, nth_up, last ) .eqv. TRUE )
      do J = nth_down, nth_up, 1
         A ( J  ) = 0
      end do
  end do
  if ( last .eqv. .TRUE.  ) then
    shared_sum = SUM
  end if
  CALL end_for ( 1 )
  END
```

Runtime Dependences

- **Variable scoping**
  - REDUCTION



```
 !$OMP PARALLEL PRIVATE( J )
       ...
C      Worksharing code
 !$OMP DO SCHEDULE( STATIC ) REDUCTION (+:SUM )
       DO J = 1, N
         SU
         A (
       END
 !$OMP EN
       ...
!$OMP EN
```

```
  SUBROUTINE th_par_code_01 (A, SUM, ...)
C Declaration of symbols
  INTEGER J, SUM
  INTEGER
   ...
C End sym
C Parallel
   ...
   CALL ws
   ...
   END
```

```
...
nth_self_p = nth_self ( )
nth_num_threads = nth_cpus_actual ( )
CALL nth_depadd ( nth_num_threads + 1 )
nth_mask = 608
do nth_p = 0,nth_num_threads-1
  nth = nth_create_vp_1s ( th_par_code_01, 0, nth_self_p, nth_p, nth_mask, 7, A, SUM,...)
end do
CALL nth_block ( )
...
```

Runtime Dependences

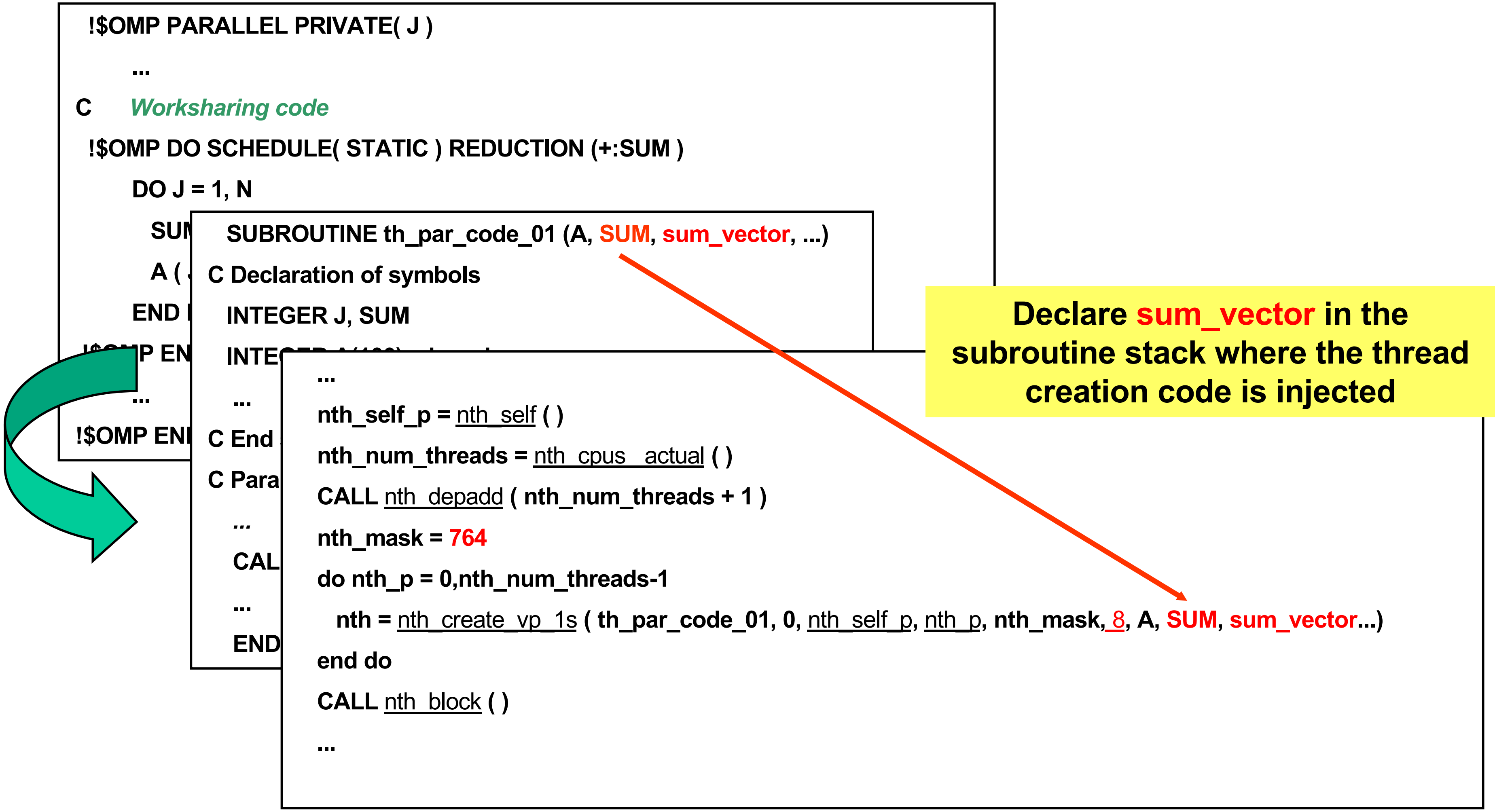
- **Variable scoping**
  - REDUCTION

```
  SUBROUTINE ws_DO_01 ( A, shared_sum, ... )
C Declaration of symbols
  INTEGER SUM
  INTEGER shared_sum
  INTEGER J
  INTEGER A(100)
  ...
C End symbol declarations
C Code for thread iterations
  SUM = "neuter of operation"
  CALL begin_for ( 1, N, 1, 01, 4 )
  do while ( next_iters ( nth_down, nth_up, nth_last ) .eqv. TRUE )
     do J = nth_down, nth_up, 1
       SUM = SUM + A ( J )
       A ( J  ) = 0
     end do
  end do
  CALL nth_spin_lock ( shared_sum )
  shared_sum = SUM
  CALL nth_spin_unlock ( shared_sum )
  CALL end_for ( 1 )
  END
```

Mutual exclusion

Runtime Dependences

- **Variable scoping**
  - REDUCTION



```
!$OMP PARALLEL PRIVATE( J )
    ...
C     Worksharing code
!$OMP DO SCHEDULE( STATIC ) REDUCTION (+:SUM )
    DO J = 1, N
      SUI
      A (
    END I
!$OMP EN
    ...
!$OMP ENI
```

```
SUBROUTINE th_par_code_01 (A, SUM, sum_vector, ...)
C Declaration of symbols
  INTEGER J, SUM
  INTE
   ...
C End
C Para
   ...
   CAL
   ...
   END
```

```
...
nth_self_p = nth_self ( )
nth_num_threads = nth_cpus__actual ( )
CALL nth_depadd ( nth_num_threads + 1 )
nth_mask = 764
do nth_p = 0,nth_num_threads-1
  nth = nth_create_vp_1s ( th_par_code_01, 0, nth_self_p, nth_p, nth_mask, 8, A, SUM, sum_vector...)
end do
CALL nth_block ( )
...
```

Declare sum_vector in the subroutine stack where the thread creation code is injected

Runtime Dependences

- **Variable scoping**
  - REDUCTION

```
  SUBROUTINE ws_DO_01 ( A, shared_sum, sum_vector,... )
C Declaration of symbols
  INTEGER SUM
  INTEGER shared_sum
  INTEGER J
  INTEGER A(100)
  ...
C End symbol declarations
C Code for thread iterations
  SUM = "neuter of operation"
  CALL nth_begin_for ( 1, N, 1, 01, 4 )
  do while ( nth_next_iters ( nth_down, nth_up, nth_last ) .eqv. TRUE )
     do J = nth_down, nth_up, 1
        SUM = SUM + A ( J )
        A ( J  ) = 0
     end do
  end do
  nth_th_id = nth_whoami ()
  sum_vector ( nth_th_id ) = SUM
  CALL nth_barrier ( )
  if ( nth_iam_master ( ) .eqv. .TRUE ) then
    do th = 0, nth_num_threads-1
       shared_sum = shared_sum + sum_vector ( th )
    end do
 end if
 CALL nth_end_for ( 1 )
  END
```

Master thread collects local computations for each thread

END