Parallel Programming with OpenMP

OpenMP directives and clauses

- Only one directive-name may be specified per directive
- OpenMP directives in Fortran begin with !$OMP
- Directives are case-insensitive.
- Directive Structure:
  - Directive name: Specifies the type of parallelism or synchronization, e.g., !$OMP PARALLEL.
  - Clauses: This is optional, modifies the behavior of the directive, such as PRIVATE, SHARED, REDUCTION, etc.

Common OpenMP directives

- !$OMP PARALLEL: Defines a parallel region where code will be executed by multiple threads.
- !$OMP DO: Distributes iterations of a loop among threads in a team.
- !$OMP SECTIONS: Defines a set of code sections to be divided among threads.
- !$OMP SINGLE: Specifies a block of code that is executed by only one thread, whichever comes first.
- !$OMP CRITICAL: Ensures that only one thread executes a block of code at a time.
- !$OMP BARRIER: Synchronizes all threads, forcing them to wait until all have reached the barrier.

End OpenMP directives

- OpenMP directives have corresponding END directives,
  - Ex: !$OMP END: !$OMP END PARALLEL
- The END directive must match the block it is ending and maintain the same structure.

Timing OpenMP routines

- OMP_GET_WTIME()
- It returns elapsed wall clock time in seconds (as double precision number)
- Example:

```
REAL(8) :: start_time, end_time
start_time = OMP_GET_WTIME()
! Code block to be timed
end_time = OMP_GET_WTIME()
write(,) 'Elapsed time (sec): ', end_time - start_time
```

OpenMP clauses

- These are optional
- PRIVATE(variable-list): Specifies that each thread has its own instance of the variables listed.
- SHARED(variable-list): Specifies that variables are shared among all threads.
- DEFAULT(PRIVATE/SHARED/NONE): Defines the default data-sharing attribute for variables.
- REDUCTION(operator): Combines variables across threads using the specified operator.

Racing condition

- A **race condition** occurs when the outcome of a program depends on the timing or sequence of uncontrollable events, like thread scheduling or the speed of each processor.
- Here, two threads are simultaneously trying to update a shared variable without proper synchronization
- Examples:
  - Two threads attempt to withdraw money from the same bank account simultaneously.
  - Multiple threads increment a shared counter variable.
  - Two threads write to the same file at the same time.

Work sharing constructs

!$OMP DO

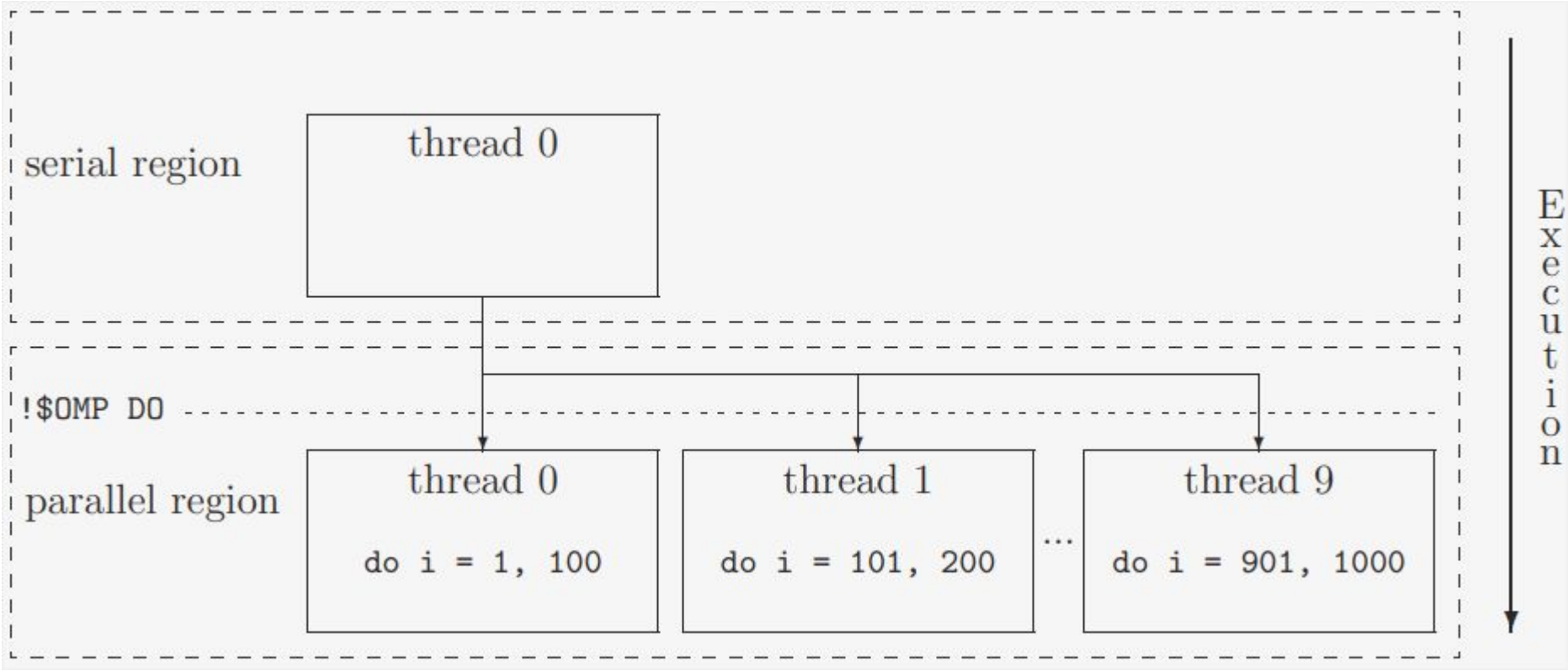
!$OMP SECTIONS

!$OMP SINGLE

!$OMP WORKSHARE

!$OMP DO

```
!$OMP DO
do i = 1, 1000
...
enddo
!$OMP END DO
```



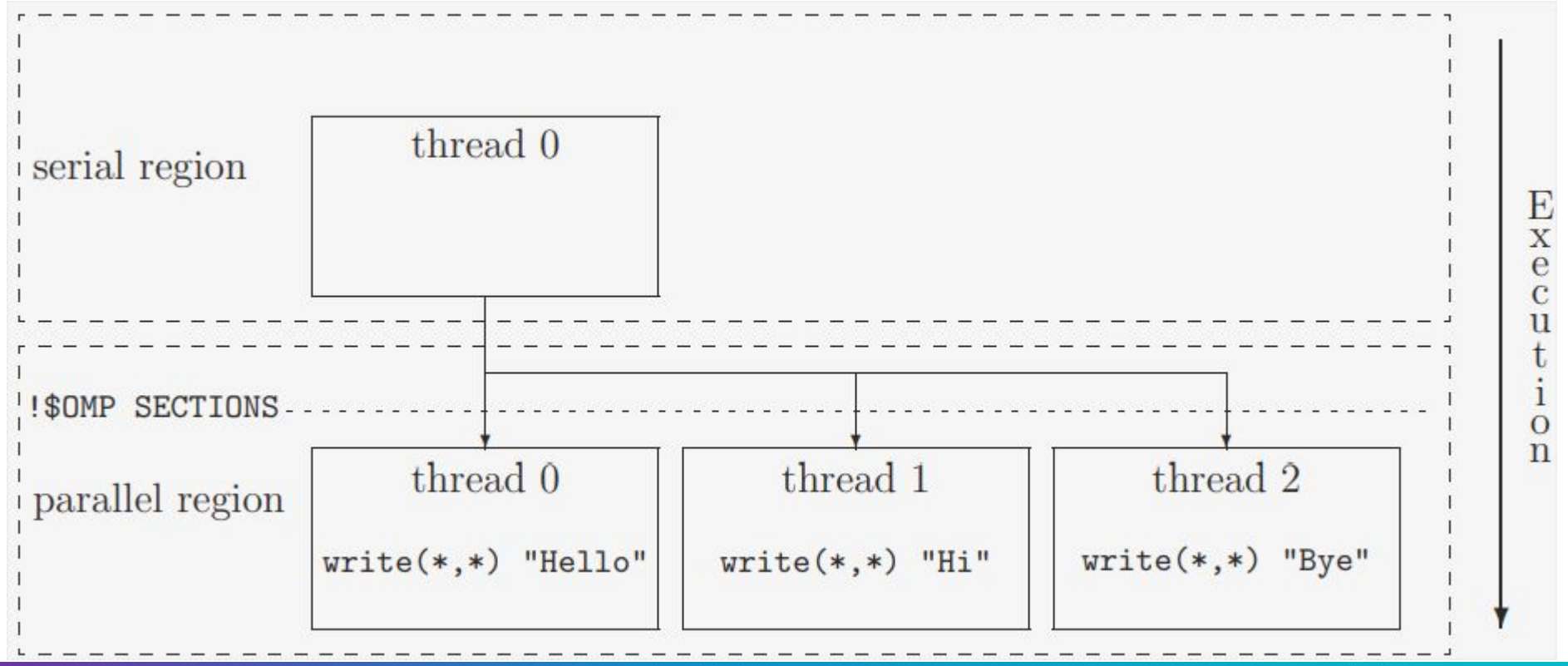
!$OMP SECTIONS

- To specify a block of code where different sections can be executed in parallel.
- Useful when you have independent tasks that can run in parallel but don't necessarily fit a loop structure.
- Syntax:

```
!$OMP SECTIONS clause1 clause2 ...
!$OMP SECTION
...
!$OMP SECTION
...
...
!$OMP END SECTIONS end_clause
```

!$OMP SECTIONS

```
!$OMP SECTIONS
!$OMP SECTION
write(*,*) "Hello"
!$OMP SECTION
write(*,*) "Hi"
!$OMP SECTION
write(*,*) "Bye"
!$OMP END SECTIONS
```



!$OMP SINGLE

- It is used to specify a block of code that should be executed by only one thread within a parallel region.
- Syntax:

```
!$OMP SINGLE
...
!$OMP END SINGLE
```

!$OMP WORKSHARE

- It is used to parallelize array operations, such as array assignments, WHERE statements, and FORALL constructs.
- Syntax

```
!$OMP PARALLEL
!$OMP WORKSHARE
   ! Array operations to be parallelized
!$OMP END WORKSHARE
!$OMP END PARALLEL
```

!$OMP WORKSHARE

```
INTEGER, PARAMETER :: n = 100
REAL :: a(n), b(n), c(n)
b=1, c=2
!$OMP PARALLEL
 !$OMP WORKSHARE
  a = b + c
  WHERE (a > 50.0)
   a = a * 2.0
  END WHERE
  FORALL (i = 1:n, a(i) > 100.0)
   a(i) = a(i) - 100.0
  END FORALL
 !$OMP END WORKSHARE
!$OMP END PARALLEL
```

Barrier and synchronization

- Some directives have an implicit barrier at the end (all threads must synchronize)
- List of directives:
  - !$OMP PARALLEL
  - !$OMP DO
  - !$OMP SECTIONS
  - !$OMP SINGLE
  - !$OMP WORKSHARE
  - !$OMP CRITICAL
  - !$OMP MASTER
  -
-

!$OMP BARRIER and NOWAIT

- !$OMP BARRIER: Explicitly synchronizes all threads at a specific point in the code
- NOWAIT: modifies the default behaviour of OMP directives, by suppressing the implicit barriers
- Examples:
  - !$OMP PARALLEL DO NOWAIT
  - !$OMP SECTIONS NOWAIT
  - !$OMP SINGLE NOWAIT

Cache coherence

- In any parallel program, ensure all threads see a consistent view of memory
- To ensure data consistency and prevent race conditions
- Each processor might have its own cache memory
- Changes made by one thread are visible to others
- Cache coherence is managed by hardware and memory architecture
- Cache coherence is not an issue in MPI

Fix the program

```
program fix
  use omp_lib
  implicit none

  integer, parameter :: n = 1000
  integer :: i, index
  integer :: x(10), y(10)
  integer :: sum_x, sum_y

  x = 0; y = 0

  !$omp parallel do reduction(+:sum_x, sum_y)
  do i = 1, n
    index = mod(i, 10) + 1

    x(index) = x(index) + 1
    y(index) = y(index) + 1
  end do
  !$omp end parallel do

  ! compute sums
  sum_x = sum(x)
  sum_y = sum(y)

  write(*,*) 'sum_x:', sum_x
  write(*,*) 'sum_y:', sum_y

end program
```

Does it give correct results? Why?

```
program test
 use omp_lib
 implicit none
 integer :: i, A(1000)

 do i=1,1000; A(i)=i; enddo

 !$omp parallel
 !$omp do
 do i = 1, 999
   A(i) = A(i+1)
 enddo
 !$omp end do
 !$omp end parallel

 do i=1,999
   if(A(i)/=i+1) write(*,*) i,A(i)
 enddo

end
```

Does it give correct results? Why?

```
program test
 use omp_lib
 implicit none
 integer :: i, A(1000)

 do i=1,1000; A(i)=i; enddo

 !$omp parallel
 !$omp do
 do i = 2, 1000
   A(i) = A(i-1)
 enddo
 !$omp end do
 !$omp end parallel

 do i=2,1000
   if(A(i)/=i-1) write(*,*) i,A(i)
 enddo

end
```

Row major vs Column major

```
program aaa
 use omp_lib
 implicit none

 integer,parameter :: n=800
 integer :: i,j,k,A(n,n,n)
 real(kind=8) :: tstart,tend

 tstart=omp_get_wtime()
 !$OMP parallel
 !$OMP DO
 do i = 1, n
  do j = 1, n
   do k = 1, n
    A(i,j,k)=i*j*k
   enddo
  enddo
 enddo
 !$OMP END DO
 !$OMP end parallel

 tend=omp_get_wtime()
 write(*,*) "Time i,j,k: ",tend-tstart
```

```
 tstart=omp_get_wtime()
 !$OMP parallel
 !$OMP DO
 do k = 1, n
  do j = 1, n
   do i = 1, n
    A(i,j,k)=i*j*k
   enddo
  enddo
 enddo
 !$OMP END DO
 !$OMP end parallel

 tend=omp_get_wtime()
 write(*,*) "Time k,j,i: ",tend-tstart

end
```

Topic: Parallel Programming with OpenMP

Reading material

- https://curc.readthedocs.io/en/latest/programming/OpenMP-Fortran.html
- https://openmp.org/wp-content/uploads/F95_OpenMPv1_v2.pdf