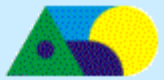
Short introduction to OpenMP

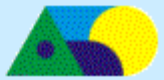
Jure Jerman, Environmental Agency of Slovenia



Outline

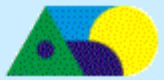
- Motivation and Introduction
- Basic OpenMP structures
- Description of exercise

Warning: far from being comprehensive introduction to OpenMP

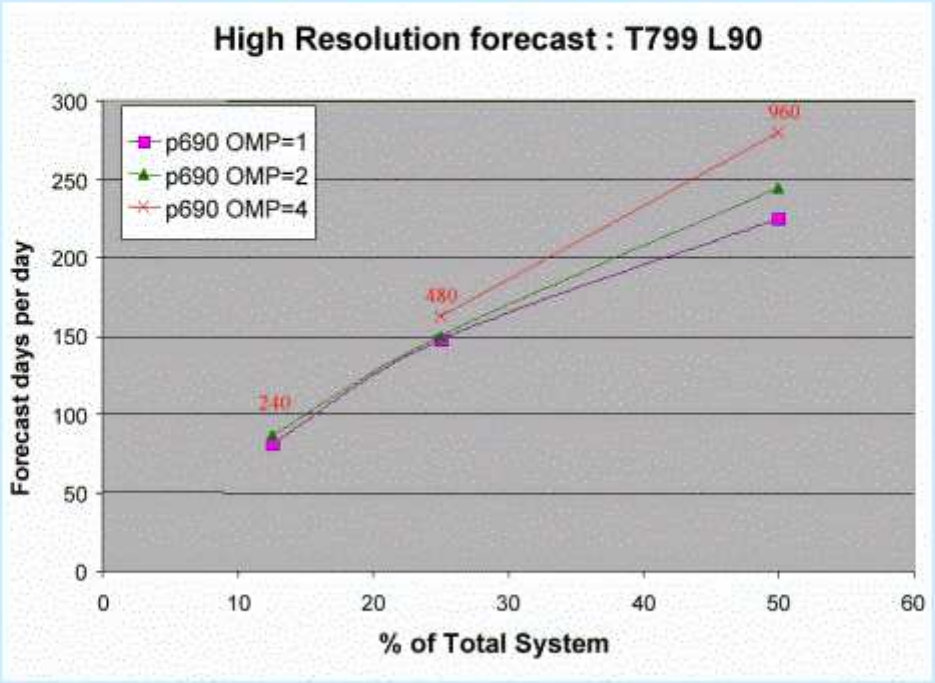


Motivation

- Two kinds of parallelization in ALADIN: MPI and OpenMP
- MPI: explicit parallelization
- OpenMP: set of compiler directives in the code
- It was believed for quite a long time, that OpenMP can not compete with MPI where programmer holds everything in hands, but:
  - with OpenMP the amount of computational overhead related to halo is reduced
  - Amount of communication is reduced as well
- => Better scalability for bigger number of processors
- The new computer architecture (SMP machines, or clusters of SMP machines)

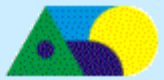


Example of OpenMP efficiency for IFS code



Introduction to OpenMP

- OpenMP: Higher level of abstraction model for parallelization set up by consortium of HPC vendors
- Consisted of compiler directives, library routines and environmental variables
- Suitable for SMP (symmetrical multi processing) computers)
- For cluster of SMPs, the communication between computers has to be done via MPI



Our first program

- OpenMP compiler directives
  - !$OMP

```
!$OMP PARALLEL DEFAULT(NONE) &
!$OMP SHARED(A,B)
```

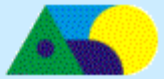
  - !$

```
!$ I = OMP_get_thread_num()
```

- Parallel region constructor

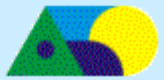
```
!$OMP PARALLEL / $OMP END PARALLEL
```

The constructs will be ignored by compiler without OpenMP.

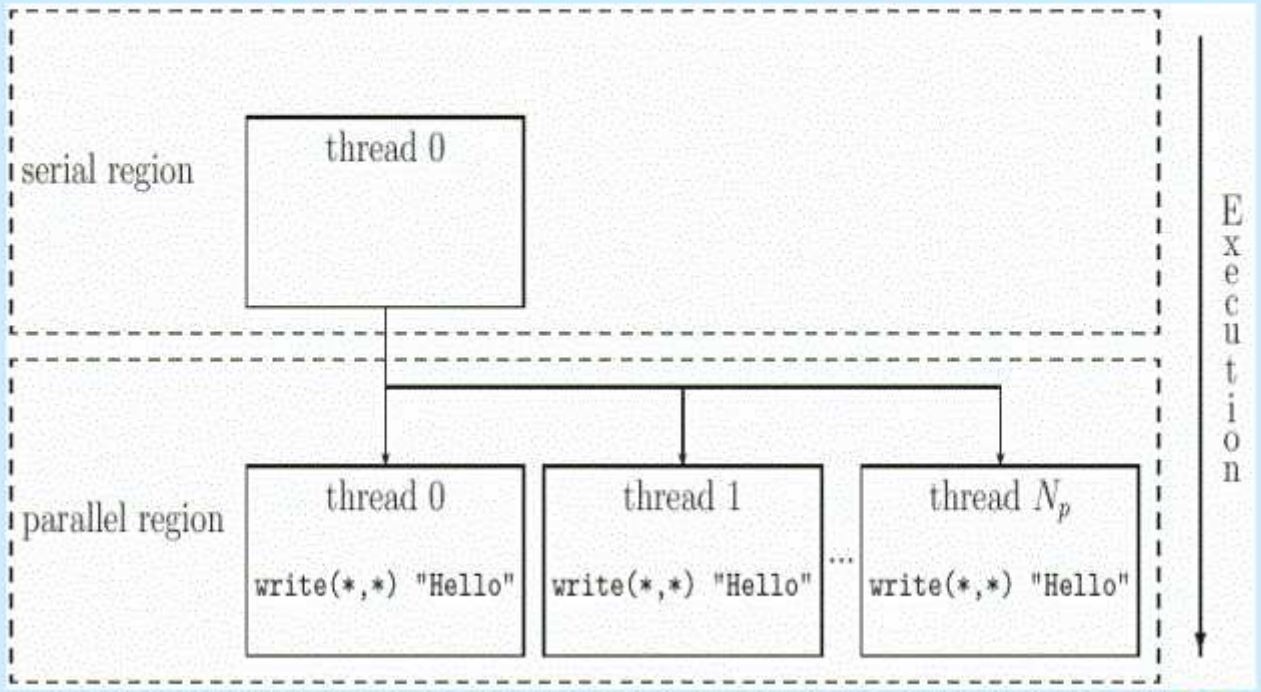


Hello world in OpenMP (1)

```
Program Hello
$!OMP PARALLEL
Write(*,*)'Hello'
$!OMP END PARALLEL
End Program Hello
```

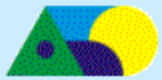
Hello world (2)



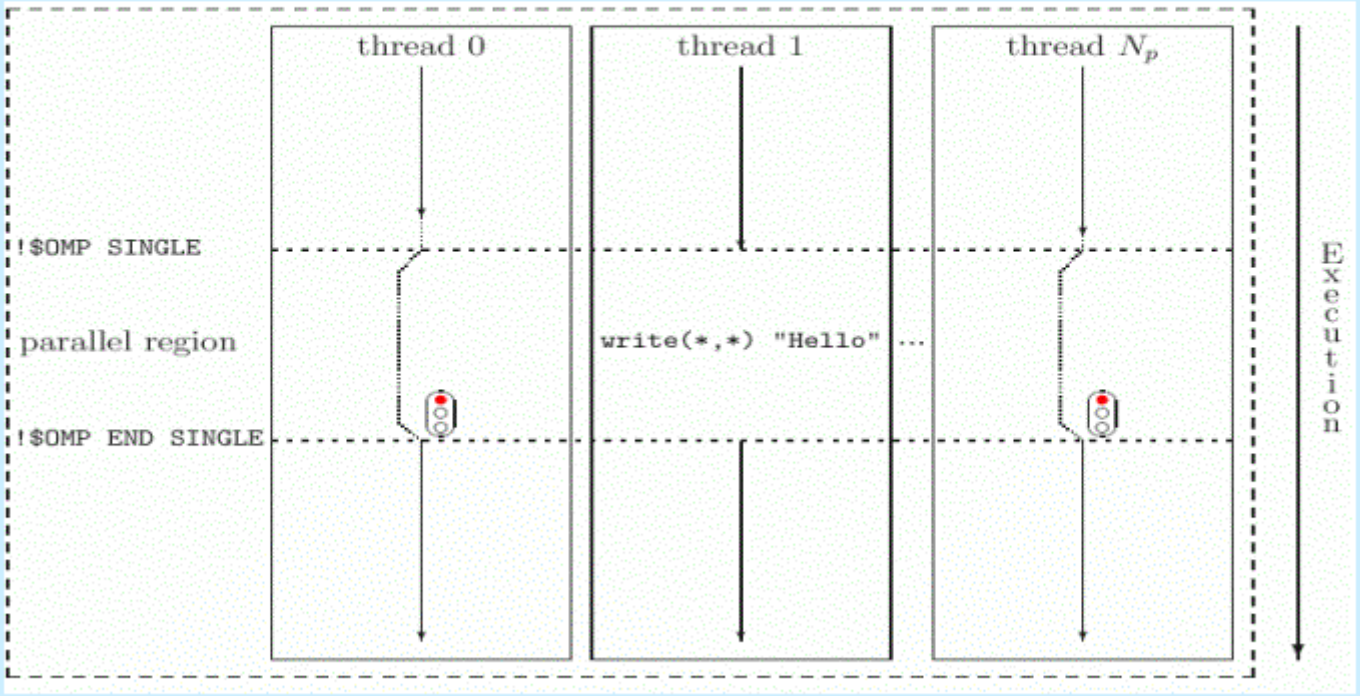
Hello world (3)

- Now we want just one thread to print on the screen:
- We use `!$OMP SINGLE` directive

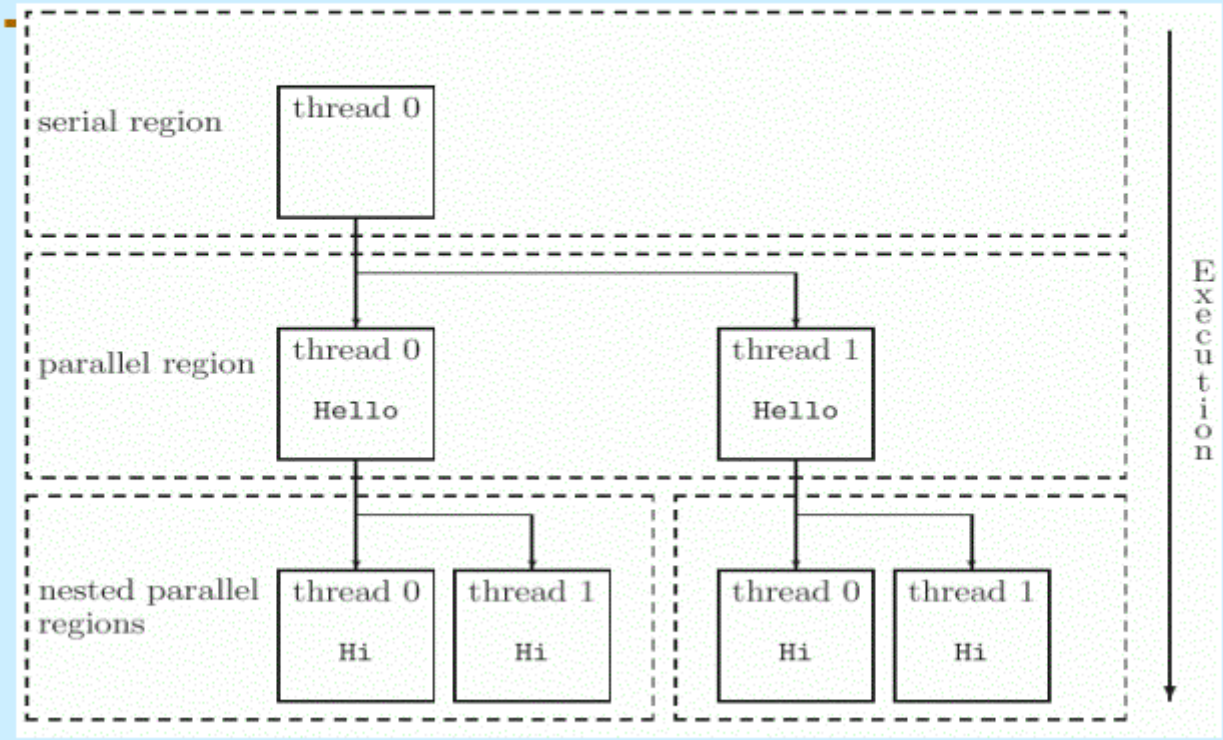
```
!$OMP SINGLE
Write(*,*) "Hello"
!$OMP END SINGLE
```

Hello world (4)



Nesting of parallel regions



Variables in parallel regions

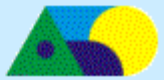
```
Program Hello
I=5
$!OMP PARALLEL
I=I*5
$!OMP END PARALLEL
Write(*,*) I
End Program Hello
```

The output for 2 threads is 125

Every thread has access to variable trough shared memory

Variables can be declared as shared or private to the threads like:

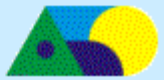
```
!$PARALLEL SHARED(A) PRIVATE(I)
```

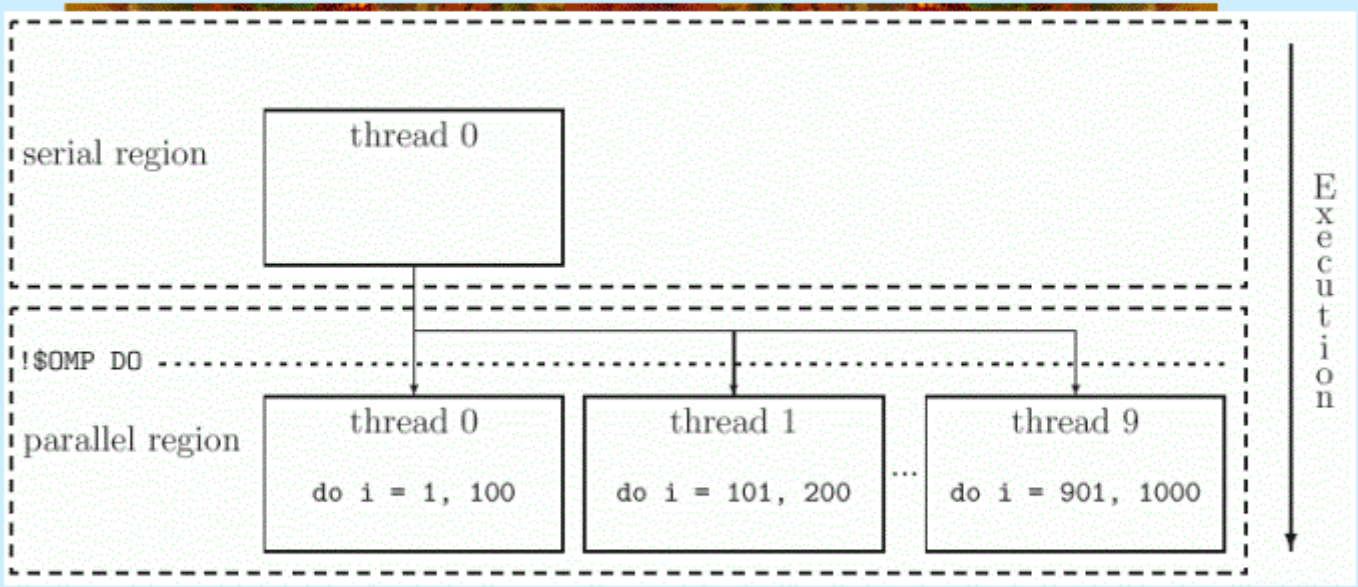
Worksharing constructs

- Purpose: To do some real parallelism
- The work sharing directives have to be put inside !$OMP PARALLEL and !$ END PARALLEL
- Best example: !$OMP DO clause/ !$OMP END DO

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

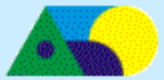



!$OMP DO / !$OMP END DO



!$OMP DO clause1, clause2, ..

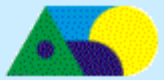
- Clause could be:
  - `PRIVATE()`
  - `ORDERED`
  - `SCHEDULE`
  - ....
- `SCHEDULE (type, chunck)`:
  - Way how to control the distribution of iterations between different threads (type can be DYNAMIC, STATIC), chunk is the portion of the loop distributed to separate threads



Parallelizing some loops might be tricky...

- Not all of loops can be made parallel:
- The chunks of loops are distributed in unpredictable way

```
REAL :: A(1000)
DO I = 1..999
  A(I)=A(I+1)
ENDDO
```

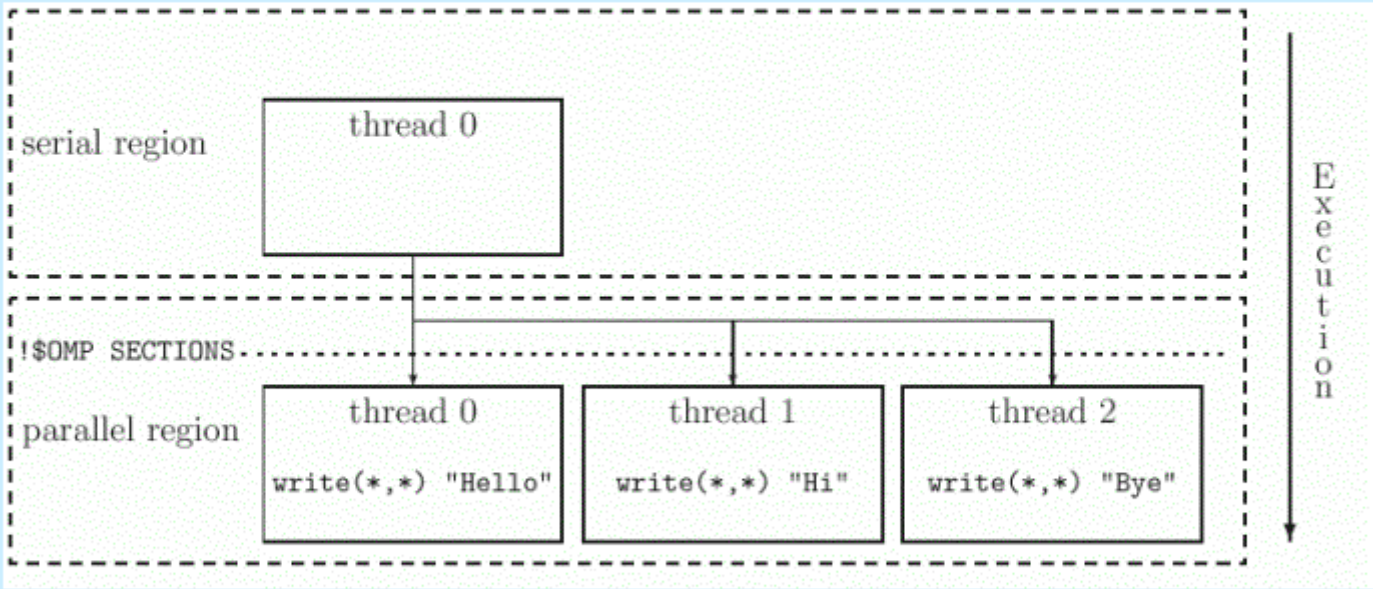



!$OMP SECTION(1)

- It is possible to create MPMD (Multiple Program, Multiple Data) style of programs with !$OMP SECTION

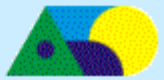
```
!$OMP SECTIONS
!$OMP SECTION
  Write(*,*)"Hello"
!$OMP SECTION
  Write(*,*)"Hi"
!$OMP SECTION
  Write(*,*)"Bye"
!$OMP END SECTIONS
```

!$OMP SECTION(2)



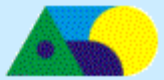
OpenMP run-time library

- OpenMP Fortran API run-time library: control and query tool for the parallel execution environment
- Set of external procedures with clearly defined interfaces delivered trough `omp_lib` Fortran module
- Main categories:
  - Execution environment routines
  - Lock routines
  - Timing routines



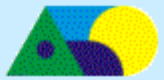
Some OMP function examples

- `OMP_get_num_threads`: number of currently used threads
- `OMP_get_thread_number`: The identification number of current thread
- Locking routines: synchronization mechanism different from OpenMP directives



The environment variables

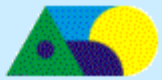
Provide control of OpenMP behavior at runtime:
`OPM_NUM_THREADS`: Number of threads to be used during execution of parallel region
`OMP_SCHEDULE:` What to do with !$OMP DO loops
`OMP_DYNAMIC` (boolean):Dynamical adjustment of number of threads by Operating System
`OMP_NESTED` (boolean): What to with nested parallelizem



OpenMP constructs in IFS/Arpege/ALADIN code

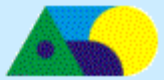
- `!$OMP PARALLEL / $OMP END PARALLEL`
- `!$OMP DO / !$OMP END DO`

```
!$OMP DO PRIVATE() SHARED()
!$OMP DO SCHEDULE(DYNAMIC,1)
```

Description of excersize

- Paralleliize the serial Fortran95 code (400 lines)
- Shallow water model with periodic LBC
- One main program, no subroutines

REFERENCES:

- http://www.openmp.org
  - OpenMP Fortran interface specification 2.0
  - Parallel Programming in Fortran 95 using OpenMP