Orchestrating a brighter world

NEC

Introduction to NEC HPF

2020/9

NEC

Table of Contents

Introduction to HPF

What is High Performance Fortran (HPF)?

- **A set of extensions to Fortran 95 for distributed-memory parallel computers**

International Standard

- by major vendors, universities, and research institutes of parallel processing

Easy to Write

- Fortran + Comment line directives

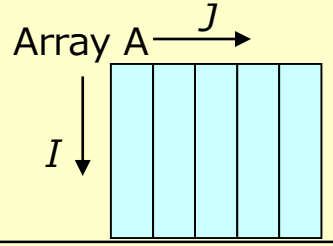
Detailed Tuning is Possible

- Control of data transfer, assignment of computation to a process

- **Ex. Insert HPF directives to serial Fortran code**

```
      DIMENSION A(10000,10000)
!HPF$ DISTRIBUTE A( *, BLOCK)
!HPF$ INDEPENDENT, NEW(I)
      DO J = 1, 10000
        DO I = 1, 10000
          A(I,J) = 0.0
        END DO
      END DO
```

Specify how arrays are distributed (data mapping)

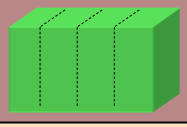


Specify the immediately following loop is parallelizable

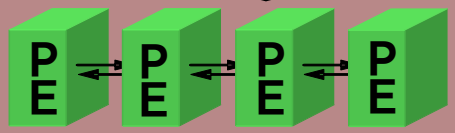
Models of Distributed-Memory Parallel Programming

Global Model

Users describe the behavior of a whole program dividing data into *n* pieces.



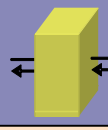
Compilers divide the program into *n* pieces.



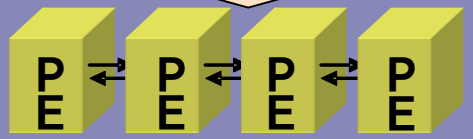
HPF

MPMD/SPMD Model (Local Model)

Users describe the behavior of a portion of a program.



Users execute *n* pieces in parallel

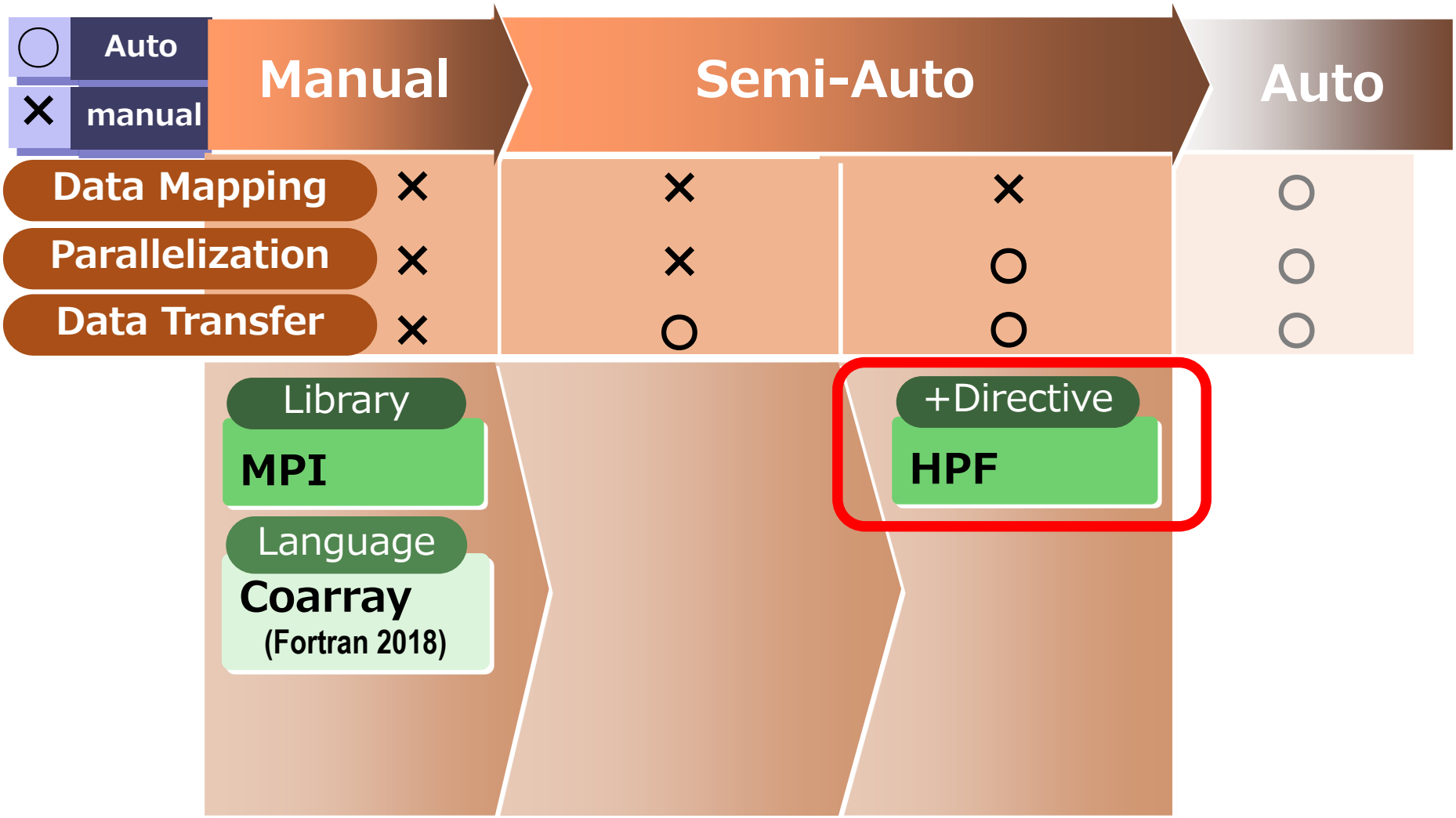


MPI, Coarray

- In regular programs, the global model is easier to parallelize and maintain

•Existing serial programs are regarded as the global model.
•Users have to understand the behavior of a whole program anyway.

Automation of Distributed-Memory Parallelization



Code Examples

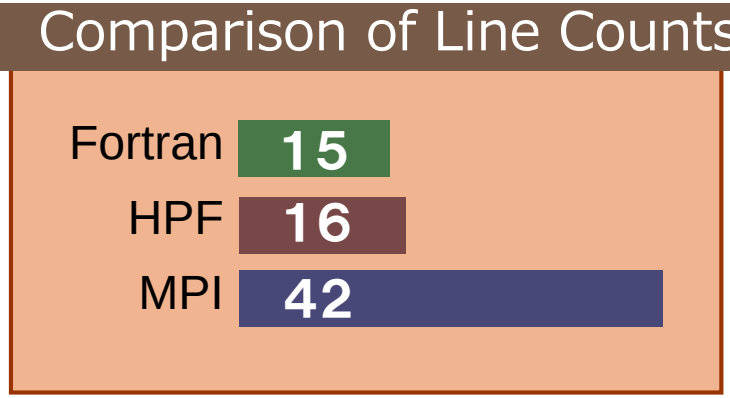
Fortran

```
parameter(n=100)
real*4 x(n),y(n)
do i=1,n
  x(i) = 1.e0 / float(i)
enddo
y = 0.e0
do i=2,n-1
   y(i) = (x(i-1)+2*x(i)+x(i+1))*0.25
enddo
dot=0.e0
do i=1,n
   dot = dot+ y(i)*y(i)
enddo
write(6,*)'dot =', dot
end
```

HPF

```
      parameter(n=100)
      real*4 x(n),y(n)
!hpf$ distribute (block) :: x,y  ! Data Mapping
      do i=1,n
        x(i) = 1.e0 / float(i)
      enddo
      y = 0.e0
      do i=2,n-1
         y(i) = (x(i-1)+2*x(i)+x(i+1))*0.25
      enddo
      dot=0.e0
      do i=1,n
         dot = dot+ y(i)*y(i)
      enddo
      write(6,*)'dot =', dot
      end
```

Comparison of Line Counts



MPI (4 Way Parallel)

```
include 'mpif.h'
 parameter(n=25)
 real*4 x(0:n+1),y(n),dotr  ! Modify declaration to add "halo"
 integer ist(mpi_status_size)
 call mpi_init(ierr)                                  ! Initialization of MPI
 call mpi_comm_size(mpi_comm_world,np,ierr)
 call mpi_comm_rank(mpi_comm_world,id,ierr)
 do i=1,n
    x(i) = 1.e0 / float(i+50*id)
 enddo
 y = 0.e0
 lb = 1                                               ! Data transfer
 ub = n
 isnd1=id+1
 ircv1=id-1
 isnd2=id-1
 ircv2=id+1
 if(id.eq.0)then
     ircv1=mpi_proc_null
     isnd2=mpi_proc_null
     lb=2
 else if(id.eq.np-1)then
     isnd1=mpi_proc_null
     ircv2=mpi_proc_null
     ub=n-1
 endif
 call mpi_sendrecv(x(n),1,mpi_real,isnd1,0,
&      x,1,mpi_real,ircv1,0, mpi_comm_world,ist,ierr)
 call mpi_sendrecv(x(1),1,mpi_real,isnd2,1,
&      x(n+1),1,mpi_real,ircv2,1, mpi_comm_world,ist,ierr)
 do i=lb,ub
    y(i) = (x(i-1)+2*x(i)+x(i+1))*0.25
 enddo
 dot=0.e0
 do i=1,n
    dot = dot+ y(i)*y(i)
 enddo
 call mpi_reduce(dot,dotr,1,mpi_real,mpi_sum,0, ! Data transfer
&  mpi_comm_world,ierr)
 if(np.eq.0)write(6,*)'dot =', dotr
 call mpi_finalize(ierr)                          ! Finalization of MPI
 end
```

Advantages of HPF

Easy to Develop and Change	Possible at Fortran (Serial) Level Changes of physical models and algorithms
Easy to Parallelize	No need to write data transfer and synchronization Just describe directives or assignments
Easy to Change Parallelization Methods	Such as from 1-dimensional distribution to 2-dim. Algorithms and parallelization are separated
Easier to Debug	Possible at Fortran (Serial) Level, basically In MPI, it is difficult to identify whether the problem is in data transfer or algorithms

Introduction to NEC HPF

Features of NEC HPF

Automatic Distributed-Memory Parallelization

- Parallelization of loops and generation of data transfer including reduction computation such as SUM and MAXLOC.
- Automatic allocation of data transfer buffer (halo) and speedup of data transfer with reuse of data transfer schedules
- Compile-time option to specify data distribution

Various Extensions are supported as well as HPF2.0

- Major HPF Approved Extensions and HPF/JA Extensions
- Unique extensions such as partial reflect

Debug and Tuning Features for HPF Programs

- Parallelization information list, diagnostic messages
- Debug options for HPF

For Advanced Users

- Tuning using MPI partially is possible
- Three levels of parallelization in the SX-Aurora TSUBASA can be used
  - Vectorization, shared-memory parallelization, and distributed-memory parallelization

Tuning Support Feature: Parallelization Information List

Loop parallelization and data transfer can be checked at the source code level

- **Users can pick up the place to tune by “greping” the marks such as “COMM:”**

Non-Parallelizable Loop (<S>)

Insertion of an **INDEPENDENT** directive may parallelize the loop if it is actually parallelizable

```
(  1  )                      subroutine sub(a,inew,iold)
(  2  )                      real a(100,100,2)
(  3  )                !HPF$ DISTRIBUTE a(*,BLOCK,*)
(  4  ) <S>---------         do j=1,100
        COMM: SFT [a] [LINO:  5  in sample.F]
(  5  ) <N>---------           do i=1,100
(  6  )  |                       a(i,j,inew)=a(i,j-1,iold)+a(i,j,iold)
(  7  )  +---------           enddo
(  8  )                      enddo
(  9  )                      end
```

Data Transfer (COMM:)

Main cause of performance degradation. Insertion of an **ON-HOME** directive may reduce data transfer

Not parallelized parallelizable loop (<N>)

Change of data mappings may parallelize the loop

Parallelization Support Feature: Data Distribution

Compile-time option to specify data distribution of arrays

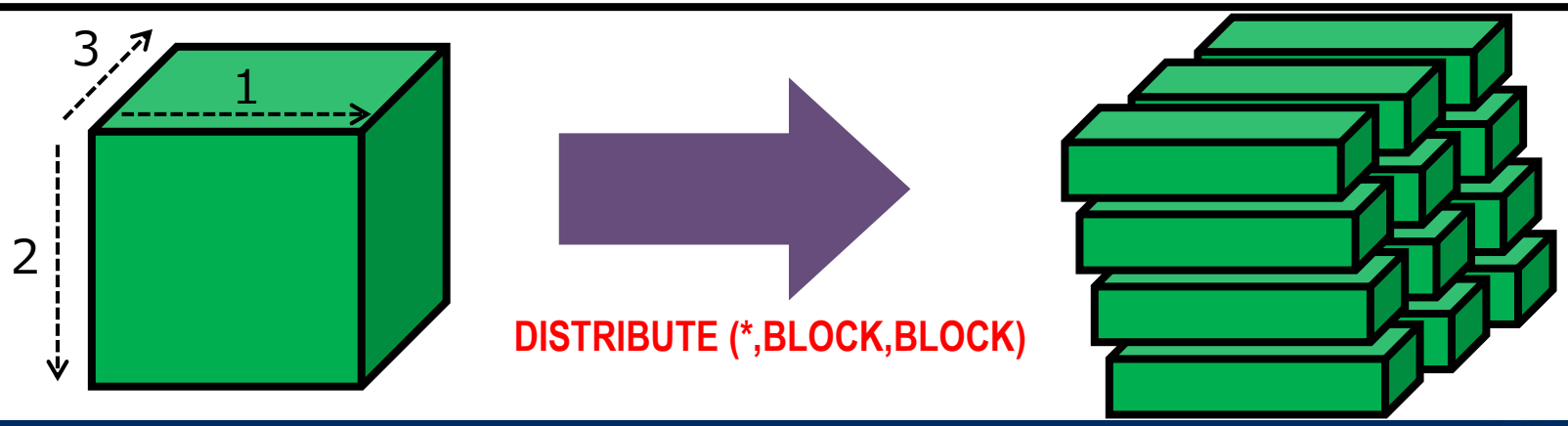
Format -Mautodist[=rank*?*[:*n*]]

- By default, all arrays are distributed with BLOCK along the last axis (except for arrays with explicit data mapping)
- =rank*?* distributes *?*-dimensional arrays with BLOCK along the last axis.
- =rank*?*:*n* distributes ?-dimensional arrays with BLOCK along the axes corresponding to 1 in binary integer *n*.

- Programs with a regular data structure may be parallelized only with this option.

Ex. Distribute 3-dimensional arrays with BLOCK along the 2nd and 3rd axes

```
%> ve-hpf -Mautodist=rank3:011  sample.hpf
```



Parallelization Support Feature: HPF Code Generation

Compile time option to generate HPF source code

Format -Mhpfout

- Generate code that has HPF directives specified with –Mautodist option

- Tuning can be started from the generated code
- Prevent mistakes such as forgetting to specify data mapping directives

Ex. **Generate code with HPF directives that distribute arrays along the last axis.**

%> ve-hpf -Mautodist -Mhpfout sample.hpf

```
      real, dimension(100,100)  :: a,b,c
!hpf$ distribute (*,*) :: a  ! Not distributed
do j=1,100
        do k=1,100
          do i=1,100
            c(i,j) = c(i,j) + a(i,k)*b(k,j)
          enddo
        enddo
      enddo
```

sample.hpf

```
      real, dimension(100,100)  :: a,b,c
!hpf$ distribute a(*,*)  ! Not distributed
!hpf$ distribute b(*,block)
!hpf$ distribute c(*,block)
      do j=1,100
        do k=1,100
          do i=1,100
            c(i,j) = c(i,j) + a(i,k)*b(k,j)
          enddo
        enddo
      enddo
```

sample.hpf.src