Practice of HPF programming

— Examples in the Earth Simulator

Earth Simulator Center

Hitoshi Murai

Table of Contents

Steps of HPF Programming

(1) Determine which dim. of arrays to be distributed.

- the last dim. of the principal array **x(nx,ny,nz)**
- the dim. of the same size **work(nz)**
- the dim. referenced in a parallel loop

(2) Insert **DISTRIBUTE** directives in each procedure.

```
!HPF$ distribute (*,*,block) :: x
!HPF$ distribute (block) :: work
```

(3) Compile and view the messages.

```
% hpf -Minfo foo.hpf
```

Steps of HPF Programming (contd.)

(4) Add an **INDEPENDENT** (+ **REDUCTION**) directive for each loop that is not shown as "**Independent loop parallelized**," if necessary.

```
5, SUM reduction generated
        1 FORALL  generated
        sum reduction inlined
```

```
!HPF$ independent, reduction(sum)
      do i=1, n
        a(i) = ...
        sum = sum + a(i)
      end do
```

```
6, Reduction call .reduce_sum emitted for variable sum
        Independent loop parallelized
```

(5) Go through trial and error to achieve acceptable high performance.

IMPACT-3D (1) Overview

- IMPlosion Analysis Code with TVD scheme (three-dimensional compressible and inviscid Eulerian fluid computation)
  - spatial differentiation: explicit 5-point stencil scheme
  - time integration: fractional time step

- Awarded the Gordon Bell Award for Language in Supercomputing 2002 [1]

※ **The source code is available in the ESC Web page for downloading.**

Example of the results of IMPACT-3D

[1] H. Sakagami, H. Murai, Y. Seo and M. Yokokawa. 14.9 TFLOPS Three-dimensional Fluid Simulation for Fusion Science with HPF on the Earth Simulator, *In proc. of SC2002*, Nov. 2002.

IMPACT-3D (2) Array Mapping

- The last dim. (i.e. 3rd dim.) of each array is distributed by **BLOCK**.
- Shadow areas are added to the distributed 3rd dimension (optional).

```
!HPF$ distribute (*,*,block) ::
!HPF$&        sr,se,sm,sp,sn,sl,
!HPF$&        walfa1,walfa2,walfa3,walfa4,walfa5,
!HPF$&        wnue1,wnue2,wnue3,wnue4,wnue5,
 ...
!HPF$ shadow (0,0,0:1) ::
!HPF$&        sr,se,sm,sp,sn,sl,
!HPF$&        wg1,wg2,wg3,wg4,wg5,
!HPF$&        wtmp1,wtmp2,wtmp3
```

IMPACT-3D (3) Loop Parallelization

- All of the loops ~~except one~~ are parallelized automatically.
- ~~An **INDEPENDENT** directive with the **REDUCTION** clause is required for parallelizing the MAX reduction as follows~~

automatically parallelized by the latest version.

```
!HPF$ independent, reduction(max:sram)
      do 10 iz = 1, lz
      do 10 iy = 1, ly
      do 10 ix = 1, lx
         wuu = sm(ix,iy,iz) / sr(ix,iy,iz)
         wvv = sn(ix,iy,iz) / sr(ix,iy,iz)
         www = sl(ix,iy,iz) / sr(ix,iy,iz)
         wcc = sqrt( sgam * sp(ix,iy,iz) / sr(ix,iy,iz) )
         sram = max( sram, abs(wuu)+wcc, abs(wvv)+wcc, abs(www)+wcc )
   10 continue
```

IMPACT-3D (4)
Vectorization and Intra-Node Parallelization

- Vectorization

  All of the loops are vectorized automatically.

  ➡ No vectorization directive is required.

- Intra-Node Parallelization

  An HPF processor is assigned to a CPU (i.e. *flat parallelization*, to be shown later).

  ➡ No microtasking directive is required.

IMPACT-3D (5) Evaluation (ver.1)

That's all !

Parallelization is completed with only **DISTRIBUTE** direc~~tives and~~ ~~one **INDEPENDENT**~~.

37 11

~~38~~ lines of ~~12~~ HPF directives in 1119 lines

For 2048x2048x4096 mesh,

➡ 12.5Tflops (38**%** of the peak)

is achieved on 4096 CPUs (8CPUs x 512PNs).

IMPACT-3D (5) Improvements

Control communications with **REFLECT and LOCAL** directives of HPF/JA extensions ➡ Communication cost reduced.

It is possible to reduce the number of message passing or schedule generation by specifying SHIFT communications explicitly.

```
!HPFJ reflect sr, sm, sp, se, sn, sl

      do iz = 1, lz-1
!HPF$ on home( sm(:,:,iz) ), local begin
      do iy = 1, ly
      do ix = 1, lx
         wu0 = sm(ix,iy,iz  ) / sr(ix,iy,iz  )
         wu1 = sm(ix,iy,iz+1) / sr(ix,iy,iz+1)
         wv0 = sn(ix,iy,iz  ) / sr(ix,iy,iz  )
         ...
```

IMPACT-3D (6) Evaluation (ver.2)

Communication control with REFLECT and LOCAL is added to the ver.1.

~~50~~ 49 lines of ~~20~~ 19 HPF directives in 1131 lines

For 2048x2048x4096 mesh,

14.9Tflops (45% of the peak)

is achieved in 4096CPUs (8CPUs x 512PNs).

Note: The MPI version achieves 15.3Tflops.

IMPACT-3D (8) Summary

- Parallelization with only **`DISTRIBUTE`** and one **`INDEPENDENT`** (ver.1) → 12.5 Tflops
- additional communication control with **`REFLECT`** and **`LOCAL`** (ver.2) → 14.9Tflops
- An HPF processor is assigned to a CPU.
- Optimal vectorization is done without directives.

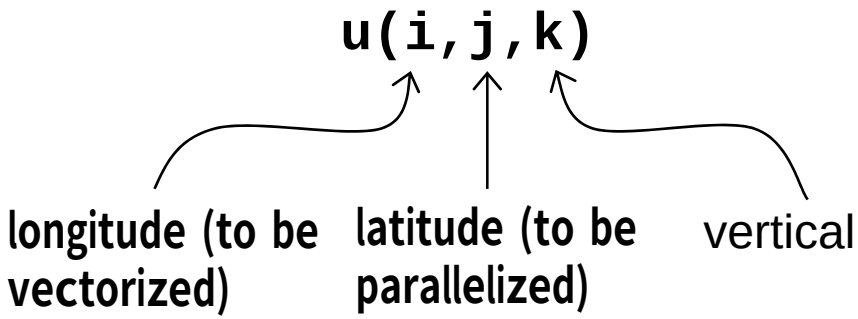
PFES (1) Overview

- Based on a numerical ocean model POM (Princeton Ocean Model) and developed for ES (POM for ES)
- A coupling code iterating computations for atmosphere and ocean by turns

PFES (2) Array Mapping

- The dim. for the latitude is distributed by **BLOCK**.
- The shadow areas are added to the distributed dim.

```
!hpf$ distribute(*,block,*) onto npr :: u
!hpf$ shadow u(0,1:1,0)
```



Distributing the last dim. is better in performance.

PFES (3) Loop Parallelization and Communication

- **INDEPENDENT**(+**REDUCTION**) and **ON**+**LOCAL** are specified for parallel loops.
- **REFLECT** directives are inserted for neighborhood accesses.

```
!HPFJ reflect d
 ...
!HPF$ independent
      do j = j2, jmx
!HPF$ on home(utf(:,j)),local begin
      do i = 2, im
        utf(i,j) = ua(i,j)*(d(i,j)+d(i-1,j))*isp2i
        vtf(i,j) = va(i,j)*(d(i,j)+d(i,j-1))*isp2i
      enddo
!HPF$ end on
      enddo
```

Most of the directives is not required for parallelization because HPF/ES can automatically parallelize the loops and generate the communications.

PFES (3) Vectorization and Intra-Node Parallelization

- Vectorization

  All of the loops are vectorized automatically.

  ➡ No vectorization directive is required.

- Intra-node Parallelization
  - The hybrid parallelization (discussed later) is applied.

    an HPF processor for a node and a microtask for a CPU
  - All of the loops are parallelized automatically.

  ➡ No microtasking directive is required.

PFES (4) Evaluation

For the resolution of 0.02 degree for the longitude and 0.025 for the latitude (18004x6002x52 mesh),

10.5Tflops (43.5**%** of the peak)

is achieved on 3008CPU (8CPUs x 376PNs).

The performance is improved to 11.11Tflops if the dims. of each array is interchanged so that the last dim. is distributed.

PFES (5) Summary

- Parallelization is done with **`DISTRIBUTE`**, **`INDEPENDENT`**(+**`REDUCTION`**), **`REFLECT`**, **`ON`**+**`LOCAL`** → 10.5 Tflops
- An HPF processor is assigned to a node and a microtask to a CPU.
- Optimal vectorization is done without directives.

3-Level Parallelism (1)

- ***Flat* Parallelization**

  An HPF processor is assigned to a CPU in each node.

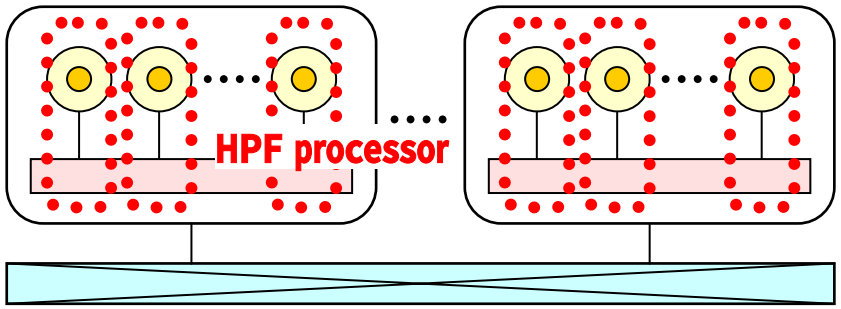
  IMPACT-3D

- ***Hybrid* Parallelization**

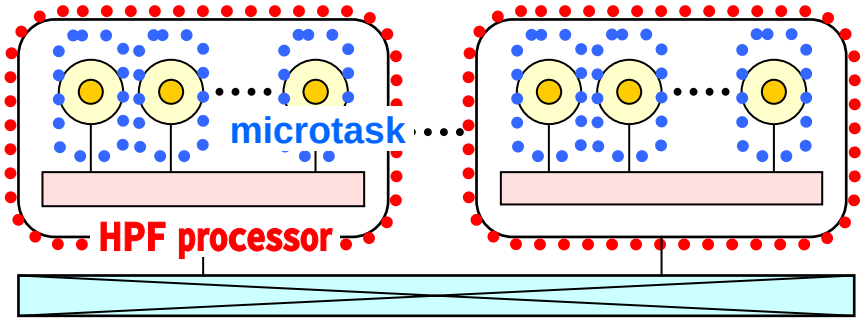
  An HPF processor is assigned to a node and intra-node parallelization is applied.

  PFES

Flat Parallelization (pure HPF)



Hybrid Parallelization (HPF+Microtasking)



3-Level Parallelism (2)

- Advantages and Disadvantages of the two Methods

	Flat	Hybrid
Peformance	low	high
Programming	easy	difficult

The difference of performance is not so large and the reverse results are possible because of the characteristics of programs. The hybrid method is superior in memory size or the performance of collective communications.

Procedure Call in a Parallel Loop

It is possible if:

- the callee procedure is *PURE*;
- the loop is *INDEPENDENT*; and
- all of the arguments are non-mapped.

```
      real a(M,N)
!HPF$ distribute (*,block) :: a

      interface
        pure subroutine sub(w)
        real w(:)
        end subroutine
      end interface

!HPF$ independent
      do j=1, N
        do i=1, M
          w(i) = a(i,j)
        end do
        call sub(w)
        do i=1, M
          a(i,j) = w(i)
        end do
      end do
```

Tips (1) Mapping

- Distribute the last dimension, if possible.

```
      real a(n,n)
!HPF$ distribute (*,block) :: a
```

```
      real a(n,n)
!HPF$ distribute (block,*) :: a
```

- Use a number for the size of the distributed dim. of each array.

```
      real a(n+1), b(n+1)
!HPF$ distribute (block) :: a, b
```

```
      real a(n), b(n+1)
!HPF$ distribute (block) :: a, b
```

- Declare the size of the distributed dim. to be multiple of #of processors.

```
      real a(1024)
!HPF$ processors p(256)
!HPF$ distribute (block) onto p :: a
```

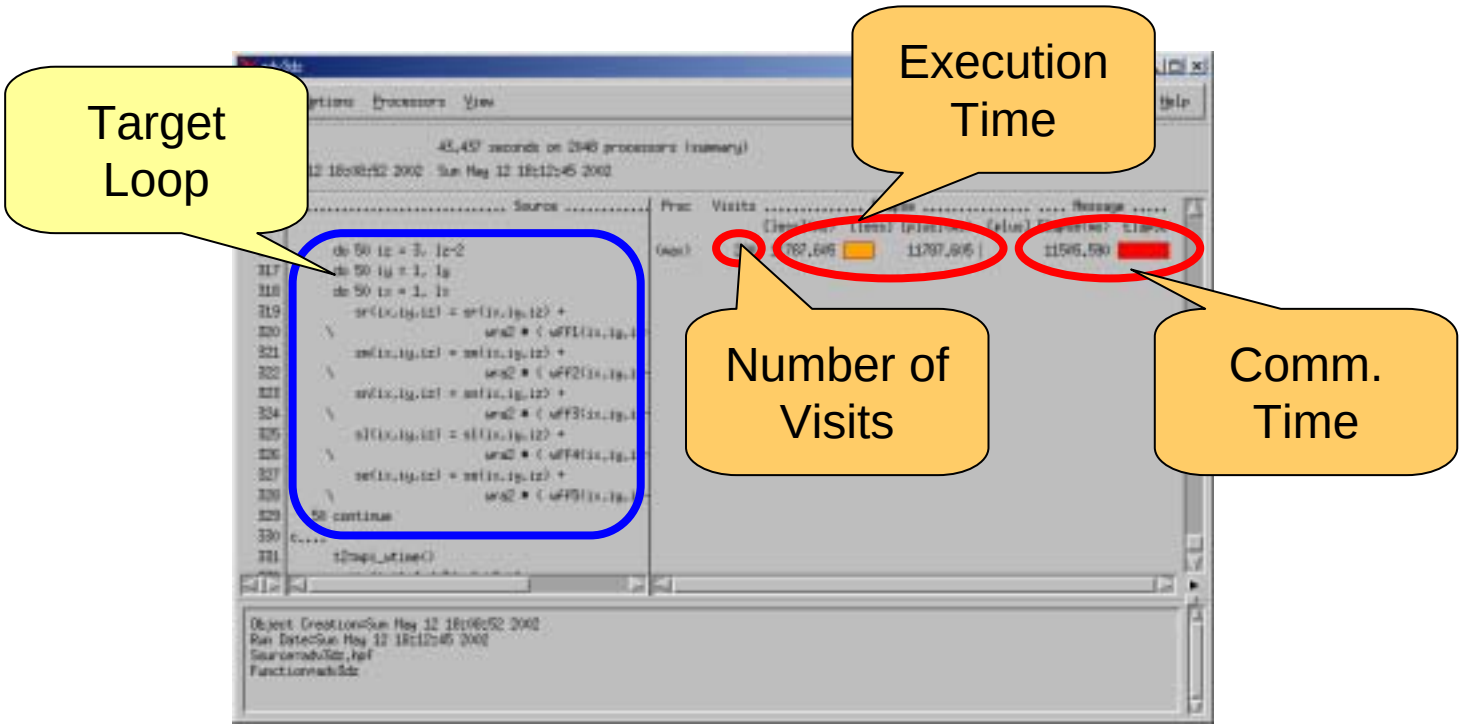
4 elements x 256CPU

```
      real a(1025)
!HPF$ processors p(256)
!HPF$ distribute (block) onto p :: a
```

5 elements x 205CPU and 0 x 51CPU

Tips (2) Tools

- HPFPROF provides variable usable information.



- MPI_PROGINF, ftrace and prof are also usable.

Tips (3) Communication Control

- Specify communications with:
  - **REFLECT**;
  - array assignment; or
  - MPI interface.
- Assert with **LOCAL** that no communication is required.

```
!HPF$ distribute (*,block) :: a, b1, c1
!HPF$ distribute (block,*) :: b2, c2

!HPFJ reflect a
      b2 = b1
      call my_transpose(c1, c2)
      ...
!HPF$ independent
      do j=1, 100
!HPF$   on home(a(i)), local begin
        ...
!HPF$   end on
      end do
```

Tips (4) Communication Optimization

- Packed Communication (message aggregation) can aggregate communications of the same pattern for different arrays into one to improve performance.

```
      real a1(100,100), a2(100,100)
     &     b1(100,100), b2(100,100)
!HPF$ distribute (*,block) :: a1, b1
      a2 = a1   ! Communication
      b2 = b1   ! Communication
```

Effective for expensive communications (i.e. those other than SHIFT)

packs the arrays into a buffer.

```
      real a1(100,100), a2(100,100)
     &     b1(100,100), b2(100,100)
!HPF$ distribute (*,block) :: a1, b1
      real buf1(2,100,100), buf2(2,100,100)
!HPF$ distribute (*,*,block) :: buf1
      buf1(1,:,:) = a1   ! Packing
      buf1(2,:,:) = b1   ! Packing
      buf2 = buf1        ! Communication
      a2 = buf2(1,:,:)   ! Unpacking
      b2 = buf2(2,:,:)   ! Unpacking
```

Tips (5) Optimization with MPI

- MPI interface

  Code region of performance bottleneck can be replaced with more efficient MPI procedures.

```
      real a(100,100), b(100,100)
!HPF$ distribute (*,block) :: a
!HPF$ distribute (block,*) :: b
      b = a  ! transposition
```

```
      real a(100,100), b(100,100)
!HPF$ distribute (*,block) :: a
!HPF$ distribute (block,*) :: b
      call my_transpose(a, b)
      ...
      end

      extrinsic(HPF_LOCAL)
     &  subroutine my_transpose(a, b)
      ...
      call MPI_send(...)
      ...
      end
```

Tips (6) I/O

- I/O of mapped arrays should be done through the parallel I/O features.
  - ∵ Normal I/O degrades performance terribly.
- Parallel files are united by the re-partioner tool and post-processed after execution.

```
% setenv HPF_FFUNIT 11
```

specify that the unit 11 is for parallel I/O

```
      real a(100)
!HPF$ distribute (block) :: a
      ...
      open(11,file='foo')
      ...
      write(11) a  ! parallel I/O
```

unite and post-process the parallel files

```
% lpionumchg -inum=N ionum=1 -f=8 foo foo2
% "post-process foo2"
```

Tips (7) Computaion mapping for boundaries

- Specifying explicitly with **ON+LOCAL** which processor to execute the computations for boundaries, improves the performance.

```
      real a(100,100)
!HPF$ distribute (*,block) :: a
      ...
!HPF$ on home(a(:,1)), local begin
      do i=1, 100
        a(i,1) = ...
      end do
!HPF$ end on
      ...
!HPF$ on home(a(:,100)), local begin
      do i=1, 100
        a(i,100) = ...
      end do
!HPF$ end on
```

Tips (8) Specifying NEW variables

- In most cases it is efficient to specify all of **NEW** variables with compiler options:
  - -Mscalarnew; and
  - -Mnomapnew,

  which reduce the task of respective specifications.

  Note: the options cannot be used and respective specifications are required if the last value is referenced after the loop.

```
!HPF$ distribute (block) :: a
      do i=1, 100
        t = a(i)
      end do
      write(*) t ! the value defined at
                 ! i=100 is referenced.
```

The option -Mscalarnew cannot be used in this case.

Tips (9) Vectorization and Intra-Node Parallelization

- Insert directives for more effective vectorization and intra-node parallelization.

  Check the compiler messages.

```
!HPF$ independent
      do k=1, nz
        do j=1, ny
          do i=1, nx
            a(i,j,k) = ...
          end do
        end do
      end do
```

The k loop parallelized by HPF is also parallelized in microtasking.

```
!HPF$ independent
!CDIR noconcur
      do k=1, nz
!CDIR   concur
        do j=1, ny
          do i=1, nx
            a(i,j,k) = ...
          end do
        end do
      end do
```

Tips (10) Calling Fortran Procedures

- In general Fortran procedures should be called as FORTRAN_LOCAL from HPF_LOCAL procedures.
- When Fortran procedures are called directly, note the following points:
  - pass the local size through an argument.
  - distribute array arguments in the last dim.
  - specify no shadow except in the last dim. of array arguments.

```
      real r(m,n), wk(m,n)
!HPF$ distribute (block,*) :: r, wk
!HPF$ shadow (0,0) :: r, wk
      interface
        extrinsic(FORTRAN_LOCAL)
     +  subroutine DFRMBF(...)
        real r(:,:)
!HPF$   distribute (block,*) :: r
        ...
        end subroutine
      end interface
      ...
      np = number_of_processors()
      call DFRMBF(n,m/np,r,m/np,1,
     +            isw,ifax,trigs,wk,ierr)
```

Tips (11) Fortran 90 Features

- Unrecommended Features
  - EQUIVALENCE Statement
  - Derived Type
  - Pointer
  - actual argument whose shape differs from that of the dummy.
  - Shared termination DO construct
  - COMMON block whose size varies with procedures
  - Assumed-size Array
  - etc.