Light HPF for PC Clusters

Hidetoshi Iwashita
<iwa@soft.fujitsu.com>
Fujitsu Limited

November 12, 2004

Background

- Fujitsu had developed HPF compiler product.
  - For VPP5000, a distributed-memory vector computer.
  - Runtime library depends on the special hardware support -- expensive DTU and barrier.
- Distributed environment is coming again.
  - Language is necessary. Data parallelism is hopeful. Techniques adopted on HPF will remain useful.
  - Architecture trend is changing dizzily and widely.
- fhpf: HPF compiler prototype
  - Aims for general distributed environment.
  - "Lightness" for users (porting, handling, etc.) & compiler (development)

Contents

- Background
- Introduction to fhpf Translator
- Technical Features
  - Normalization of mapping
  - Index localization
- Brief Evaluation
- Summary

Compilation and Execution

(1) HPF compilation

% fhpf a.hpf b.hpf

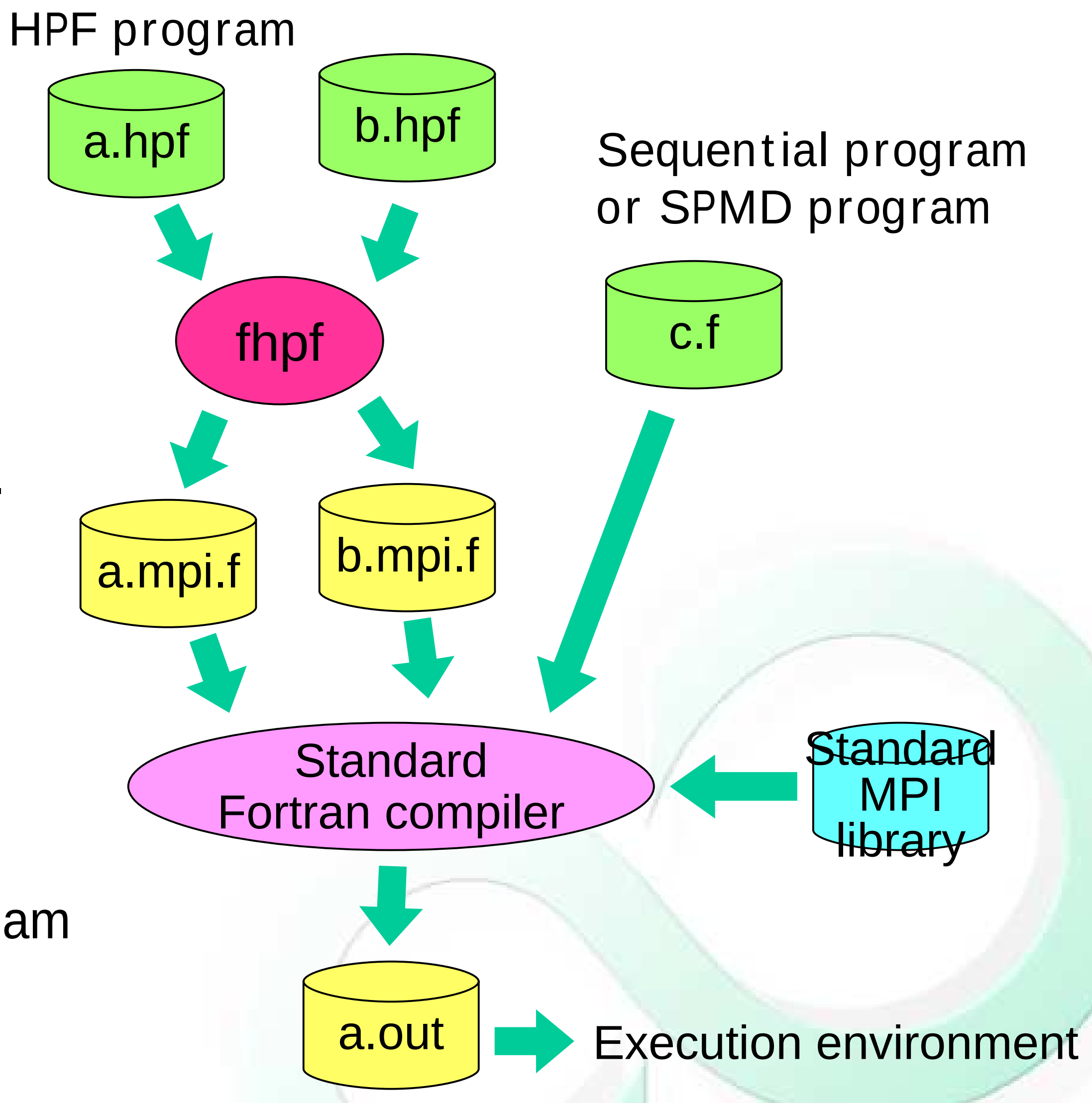
Fortran/MPI program *.mpi.f
will be generated (in default).

(2) Fortran/MPI compilation

% mpif77 a.mpi.f b.mpi.f c.f

(3) Execution as a MPI program

% mpirun -np 4 a.out



Practical Example

Input file: block.hpf

```
      integer A(100)
!hpf$ processors P(4)
!hpf$ distribute A(block) onto P

!hpf$ independent
      do i=n1,n2
        A(i)=i
      enddo
      end
```

Command lines

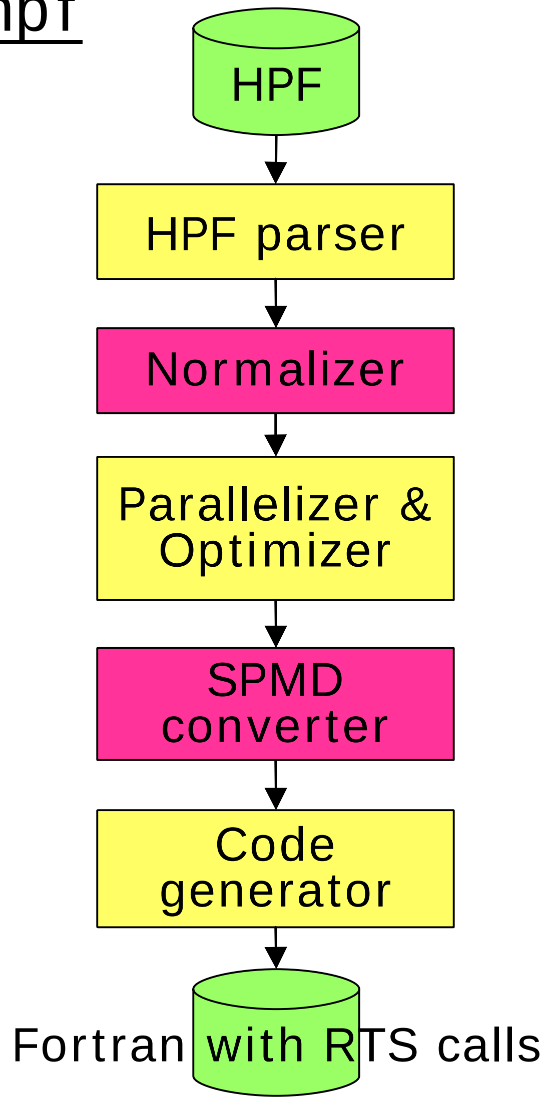
```
% fhpf block.hpf -f77
fhpf V1.2/MPI -- HPF translator for Linux system
block.hpf -> block.mpi.f
%
```

Output file: block.mpi.f

```
PROGRAM ORG__MAIN___
INCLUDE 'mpif.h'
EXTERNAL ORG___MAIN___
INTEGER*4 ORG_RANK
INTEGER*4 ORG_SIZE
INTEGER*4 ierrX0
COMMON /ORG_COMMON/ORG_RANK,ORG_SIZE
CALL mpi_init(ierrX0)
CALL mpi_comm_rank(MPI_COMM_WORLD,ORG_RANK,ierrX0)
CALL mpi_comm_size(MPI_COMM_WORLD,ORG_SIZE,ierrX0)
CALL ORG___MAIN___()
CALL mpi_finalize(ierrX0)
END
SUBROUTINE ORG___MAIN___()
INCLUDE 'mpif.h'
INTEGER*4 a(0:24)
INTEGER*4 i
INTEGER*4 n1
INTEGER*4 n2
INTEGER*4 ORG_RANK
INTEGER*4 ORG_SIZE
INTEGER*4 spmdX0
INTEGER*4 spmdX1
INTEGER*4 spmd_startX0
INTEGER*4 spmd_endX0
INTEGER*4 spmd_stepX0
INTEGER*4 ierrX0
COMMON /ORG_COMMON/ORG_RANK,ORG_SIZE
spmdX0 = mod(ORG_RANK,4)
spmdX1 = 25*spmdX0
IF (spmdX0.LE.(n1-1)/25) THEN
  spmd_startX0 = n1-1-spmdX1
ELSE
  spmd_startX0 = 0
ENDIF
IF (spmdX0.LT.(n2-1)/25) THEN
  spmd_endX0 = 24
ELSE
  spmd_endX0 = n2-1-spmdX1
ENDIF
spmd_stepX0 = 1
DO i=spmd_startX0,spmd_endX0,1
  a(i) = i+spmdX1+1
ENDDO
END
```


Technical Features

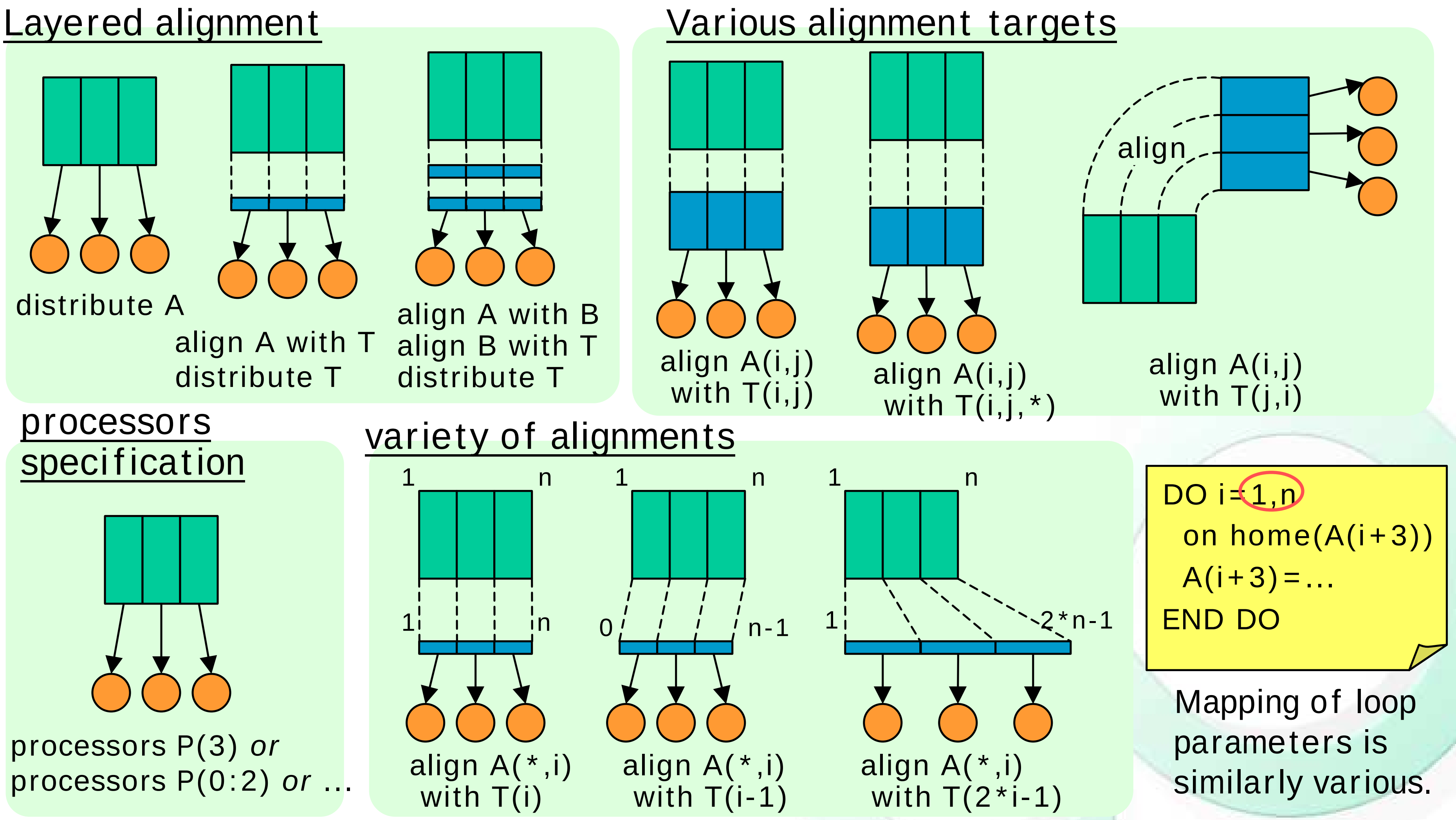
fhpf



1. Normalization of Mapping
   - Reduces variety of mapping descriptions into a standard form.
2. Index Localization
   - Part of SPMD converter
   - Global-to-local conversion of loop boundaries

Technical Features 1.
Normalization of Mapping -- Purpose

- Reduce too much variation, even for the same mapping:



Technical Features 1.

Normalization of Mapping -- Process

(1) normalize the shape of processors

(2) normalize the mapping of distributees

(3) normalize alignments

(3) unify the templates (if not dynamic)

(4) replace the home variable with the template

(4) adjust the loop parameters

Input Code

```
!hpf$ processors P(4)
      real A(1:30),C(1:29),D(1:15)
      real F(1:10,1:30)
!hpf$ template T(30)
!hpf$ distribute A(block) onto P
!hpf$ distribute T(block) onto P
!hpf$ align C(I) with A(I+1)
!hpf$ align D(I) with A(2*I)
!hpf$ align F(*,I) with T(I)

!hpf$ independent
  L1: do I=2,29
!hpf$  on home(A(I)) begin
        A(I)=C(I)*I+F(I,I)
!hpf$  end on
  end do

!hpf$ independent
  L2: do K=1,10
!hpf$  on home(D(K)) begin
        D(K)=A(2*K)
!hpf$  end on
  end do

  end
```

(3) (4)

```
A(I-1)=C(I)*I+F(I,I-1)
```

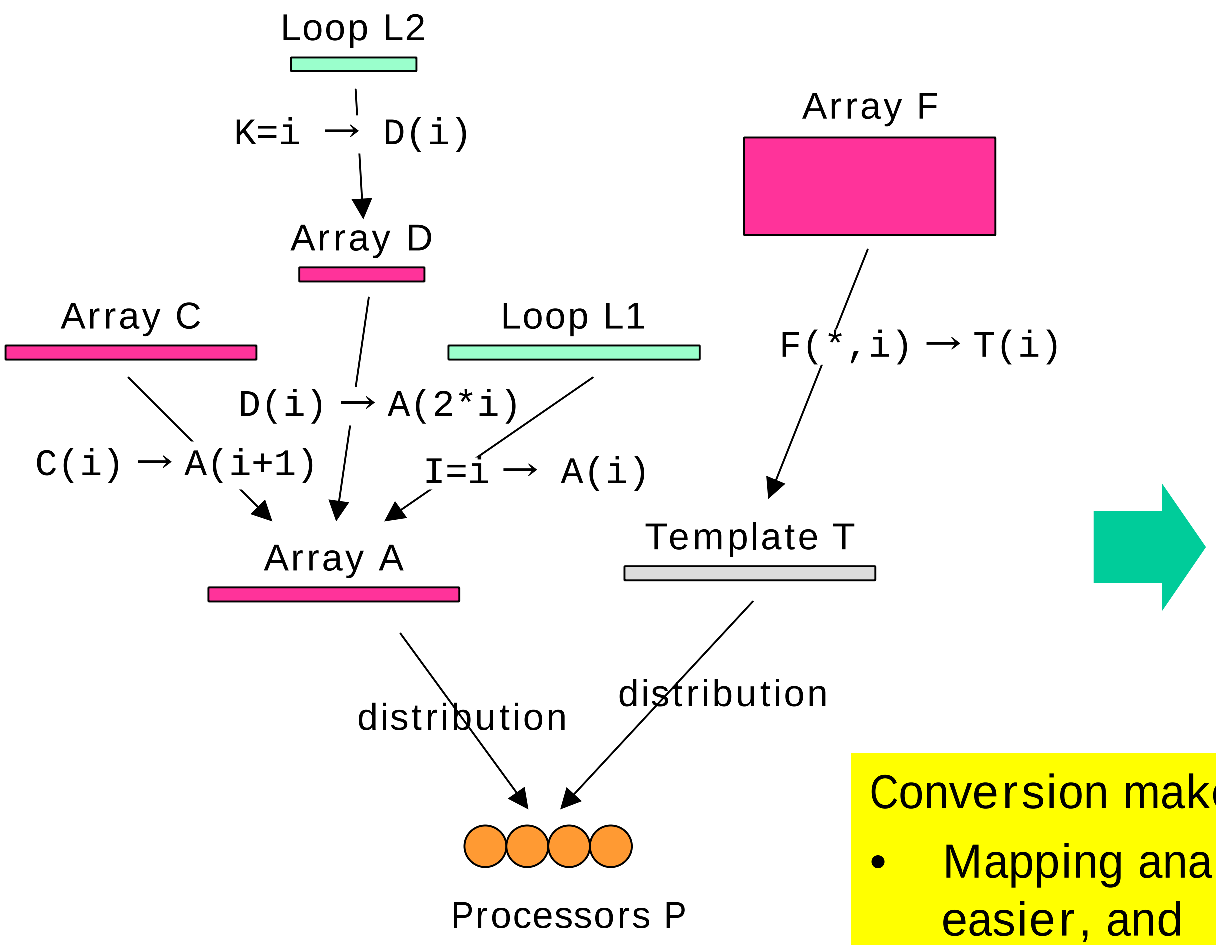
(3) (4)

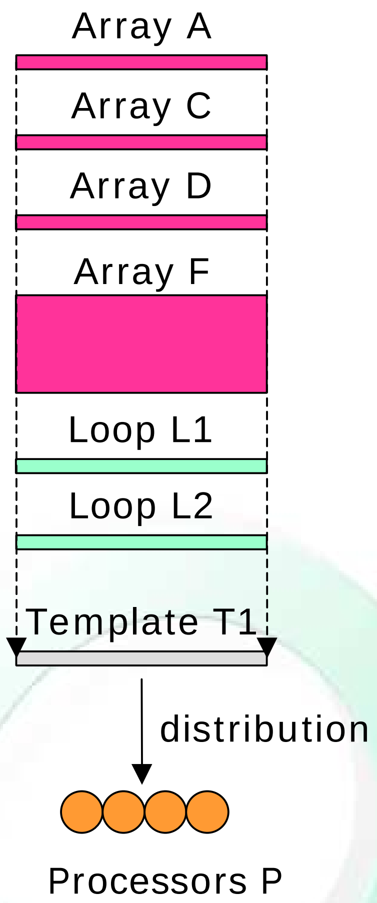
```
D(2*K-1)=A(2*K-1)
```

Normalized Code Image

```
!hpf$ processors P(0:3)
      real A(0:29),C(0:29),D(0:29)
      real F(1:10,0:29)
!hpf$ template T1(0:29)
!hpf$ distribute T1(block(8)) onto P
!hpf$ align A(I) with T1(I)
!hpf$ align C(I) with T1(I)
!hpf$ align D(I) with T1(I)
!hpf$ align F(*,I) with T1(I)

!hpf$ independent
  L1: do I=1,28
!hpf$  on home(T1(I)) begin
        A(I)=C(I+1)*(I+1)+F(I+1,I)
!hpf$  end on
  end do

!hpf$ independent
  L2: do K=1,19,2
!hpf$  on home(T1(K)) begin
        D(K)=A(K)
!hpf$  end on
  end do

  end
```

Technical Features 1.
Normalization of Mapping -- Result





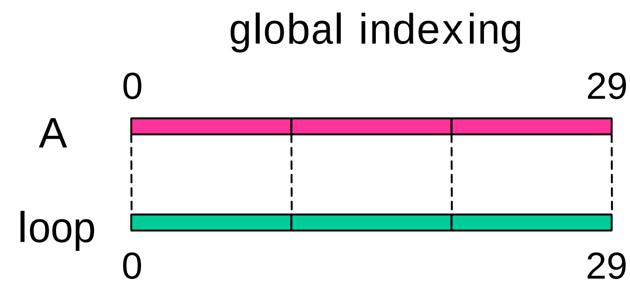
Conversion makes:

- Mapping analysis easier, and
- Following compiler modules simpler.

Technical Features 2.
Index Localization -- Purpose

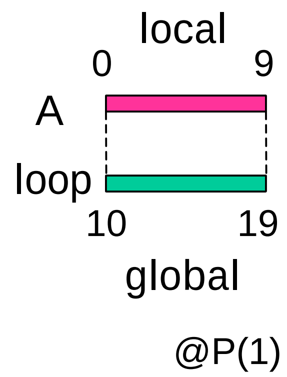
- Array subscripts keep simple in SPMD conversion.

```
!hpf$ processors P(0:2)
      real A(0:29)
!hpf$ distribute A(block) onto P

!hpf$ independent
      do I=0,29
!hpf$   on home(A(I))
        A(I)=I
      end do
```



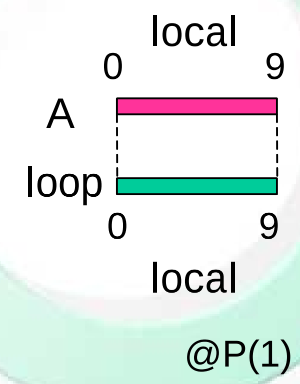
old version

```
real a(0:9)

myID=one of {0,1,2}
myLB=10*myID
myUB=myLB+9
do I=myLB,myUB
  A(I-myLB)=I
end do
```



current version

```
real a(0:9)

myID=one of {0,1,2}
myLB=10*myID
myUB=myLB+9
do i=0,9
  A(i)=i+myLB
end do
```



Technical Features 2.
Index Localization -- General Case

do I=I1,I2,I3 *on home* **A(***f***(I))**
  A(*f***(I)) = B(***g***(I))**
end do

↓

do i ∈ $gtol3_A$(**I1:I2:I3**)
  I = $ltog_A$(**i**)
  A(i) = B($gtol_B$(g(**I**))**)**@$gtop_B$(g(**I**))
end do

$gtol_X$(K) local index corresponding to global index K along the alignment of X

$ltog_X$(k) global index corresponding to local index k along the alignment of X

$gtop_X$(K) processor ID corresponding to global index K along the alignment of X

$gtol2_X$(K1:K2) , $gtol3_X$(K1:K2:K3)
  $gtol_X$ for duplet/triplet set of indices

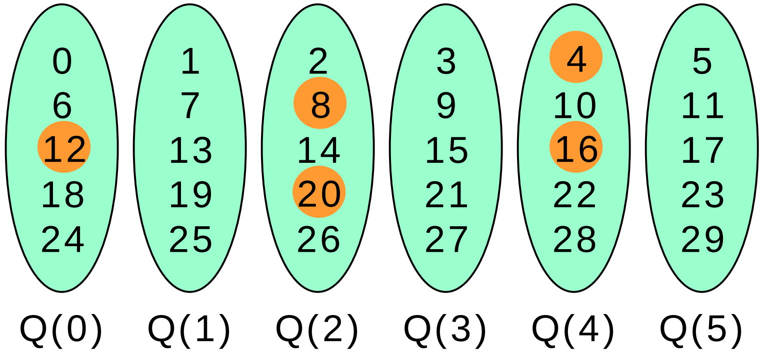
distribution	***function***			
	gtol(I)	***ltog***(i)	(i1:i2) = ***gtol2***(I1:I2)	(i1:i2:i3) = ***gtol3***(I1:I2:I3)
block(w)	I - p*w *or* mod(I, w)	i + p*w	*Available*	*Available*
gen_block(W)	I - sum(W(0:p-1))	i + sum(W(0:p-1))		
cyclic(1)	⌊I / P⌋	P*i + p		*Available in subprogram.*
cyclic(w)	w*⌊I / (P*w)⌋ + mod(I, w)	P*w*⌊i / w⌋ + p*w + mod(i, w)		*Not available in the form of triplet.*
indirect(M)	*Use a table GTOL.*	*Use a table LTOG.*	*Use a table GTOL*	

p : my processor ID, P : number of processors

Index Localization -- Ex. Cyclic/stride

Input Code (HPF)

```
      real X(100,100,0:23)
!hpf$ processors Q(0:5)
!hpf$ distribute X(*,*,cyclic) onto Q

!hpf$ independent, on home(X(:,:,k))
      do k=4,20,4
        do j=1,100
           do i=1,100
             X(i,j,k)=i*j*k
           end do
        end do
      end do
      end
```

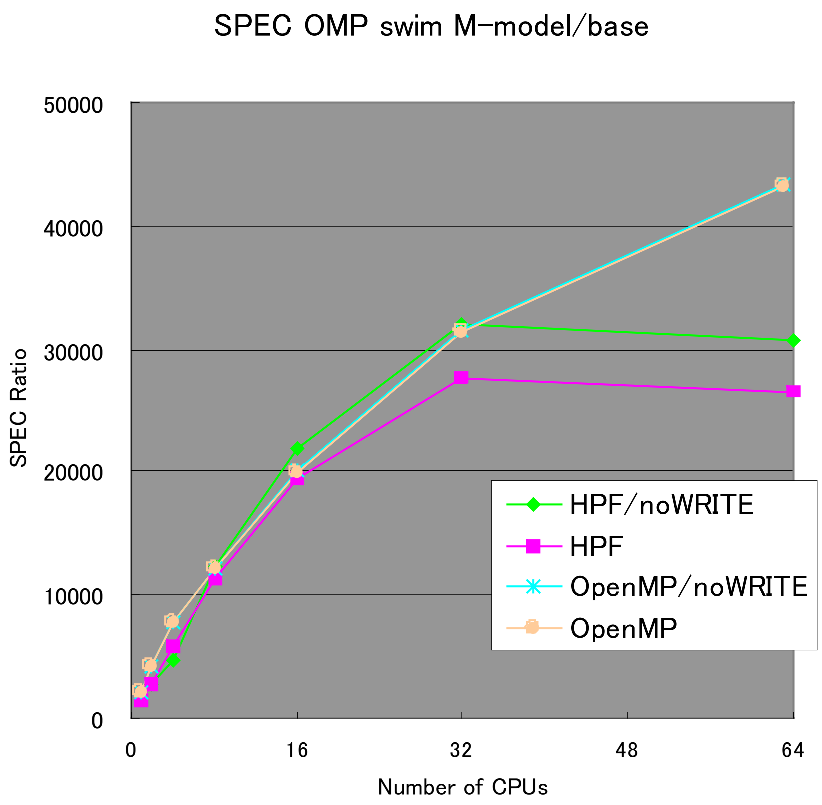


Output Code (Fortran)

```
REAL x(1:100,1:100,0:3)
INTEGER i,j,k
INTEGER myID
INTEGER k1,k2,k3
INTEGER n_cyc
INTEGER,EXTERNAL:: cyclic_f
CALL mpi_comm_rank(..., myID,...)
n_cyc = cyclic_f(4,4,6,myID,2)
IF (n_cyc.NE.-1) THEN
  k1 = (4+n_cyc*4-myID)/6
  k2 = (26-myID)/6-1
  k3 = 2
  DO k=k1,k2,k3
    DO j=1,100,1
      DO i=1,100,1
        x(i,j,k) = i*j*(6*k+myID)
      ENDDO
    ENDDO
  ENDDO
ENDIF
END

INTEGER FUNCTION cyclic_f(i1,i3,P,myID,t)
...

END FUNCTION
```

Evaluation: Comparison with OpenMP

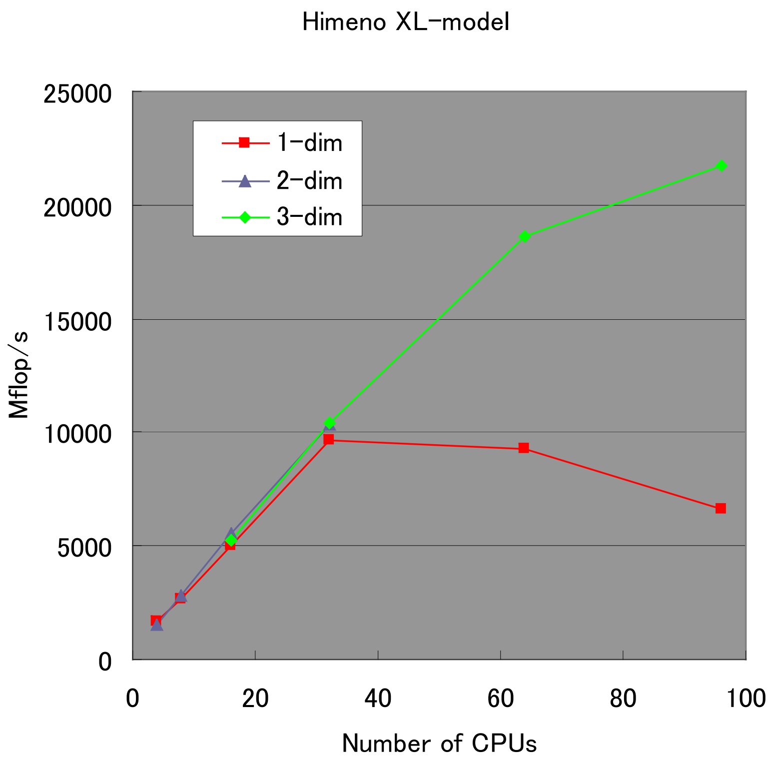
SPEC OMP2001 M-model/base
On Fujitsu PRIMEPOWER HPC2500 1.5GHz (In SMP node)
- OpenMP: Fujitsu Parallelnavi Fortran2.3
- HPF: fhpf V1.1.3, Fujitsu Parallelnavi Fortran2.3, & MPI6.1



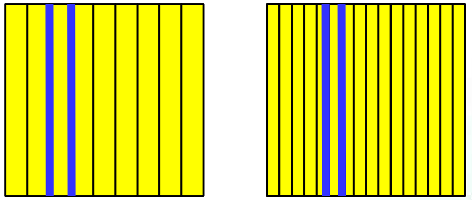
- Sometimes HPF is better than OpenMP.
- Output of distributed data causes large overhead.
- Different algorithm is required for more # of procs.
  -> multidimensional dist.

Evaluation: Multi-dimensional Distribution

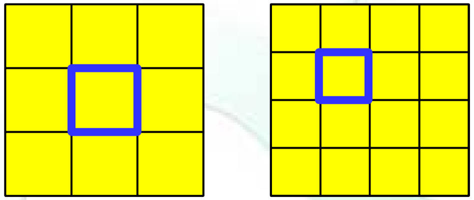
Himeno benchmark XL-model (1024x512x512 grid)
(http://w3cic.riken.go.jp/HPC/HimenoBMT/)
On Fujitsu PRIMEPOWER HPC2500 1.5GHz (In SMP node)
fhpf V1.2



- Order of shadow comm. cost:
  - O(1) for 1-dim. distribution



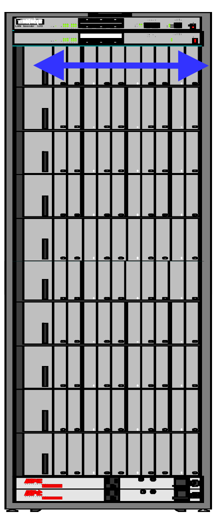
  - O(p^(-1/2)) for 2-dim.



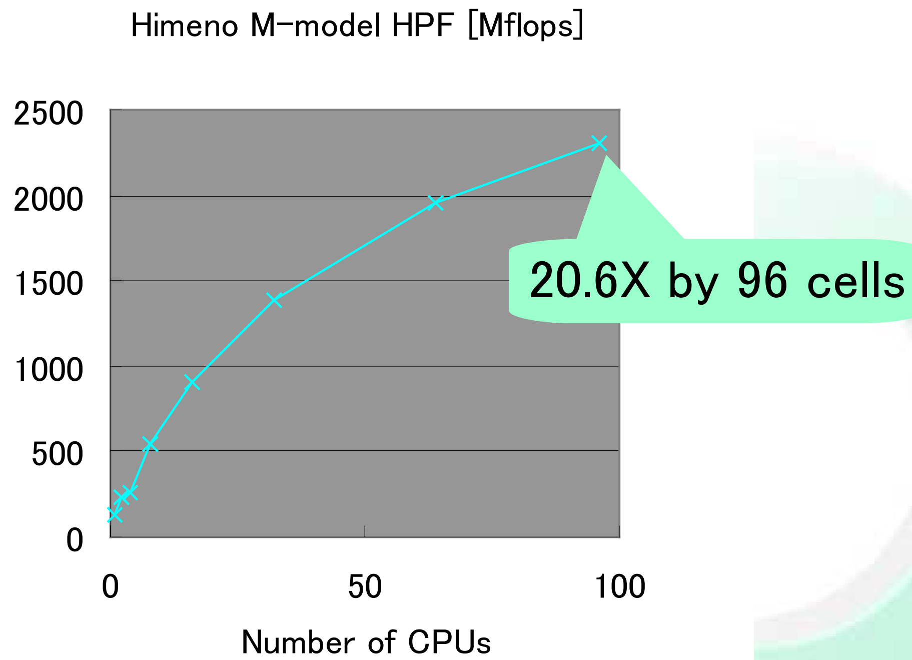
  - O(p^(-2/3)) for 3-dim.

Evaluation on Blade Server

Himeno benchmark M-model
Linux blade server
fhpf V1.0, g77, LAM-MPI



- Mobile Pentium III
- 10 blades/rack
- 1Gbit interconnect



Summary

- Debeloping fhpf Compiler
  - Key: Lightness
    - Small & simple translator suitable for distribution
    - Portablility: any environment. Only standard MPI & Fortran are required.
- Compiler Techniques
  - Normalization
    - Reduces the huge variation of mapping
  - Index localization
    - Makes loop indices local and then simplifies subscripts of vars.