Optimisations in the High Performance Fortran Compilation System ADAPTOR

Dr. Thomas Brandes

SCAI.LAB, GMD, St. Augustin

e-mail: brandes@gmd.de

October 1995

HPF Workshop, Southampton

1. ADAPTOR : HPF Compilation System
2. Compilation Strategy
3. Optimisation Strategies
4. Some Results
5. Conclusions

Research Work about HPF at GMD

HPF Language Definition

- Member of HPF core group
- Chair of European Working Group on HPF
- PACT (Proposal for Mapping), from ESPRIT project GENESIS

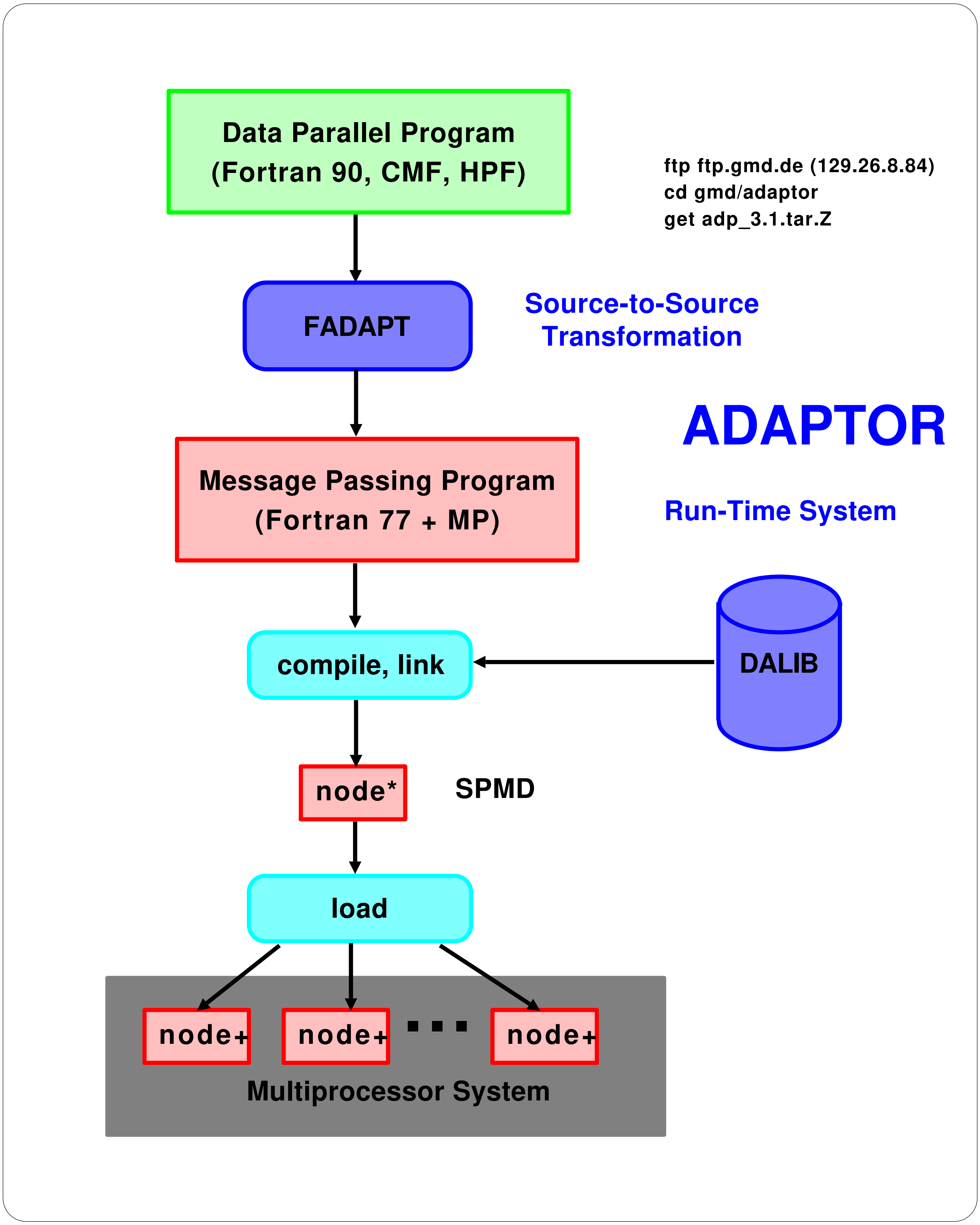
HPF Compilation System ADAPTOR

- Public Domain Tool for Compiling HPF to Fortran 77 + message passing

- Development of Optimization Strategies for HPF compilers (funded by ESPRIT project PPPE)

Porting Codes to HPF

- AVL Fire Code (within PPPE)
- ECMWF Integrated Forecasting System, 2D (within PPPE)
- cooperation with Portland Group
- providing HPF expertise within new ESPRIT project PHAROS



ADAPTOR : System Overview

ADAPTOR available on:

- Intel Paragon
- Meiko CS-2
- IBM SP-2
- Power PC
- CM-5
- HP, IBM, SUN workstations
- NEC Cenju-3

Supported communication protocols:

- PVM
- MPI
- PARMACS
- Intel NX
- Meiko ELAN
- P4
- IBM EUI

Versions:

1.0	March 1993
2.0	March 1994
3.0	June 1995
3.1	October 1995

More Information:

http://www.gmd.de/SCAI/lab/adaptor/adaptor_home.html

Supported Machines / Communication Protocols

Version 3.0 (June 1995)

- implicit redistributions at subroutine boundaries

- less restrictions for using distributed arrays
  (temporary arrays are created)

- HPF compiler optimizations

- more portability between ADAPTOR and other HPF compilers

Note: internal array descriptors are now used at runtime

Comparison to HPF subset:

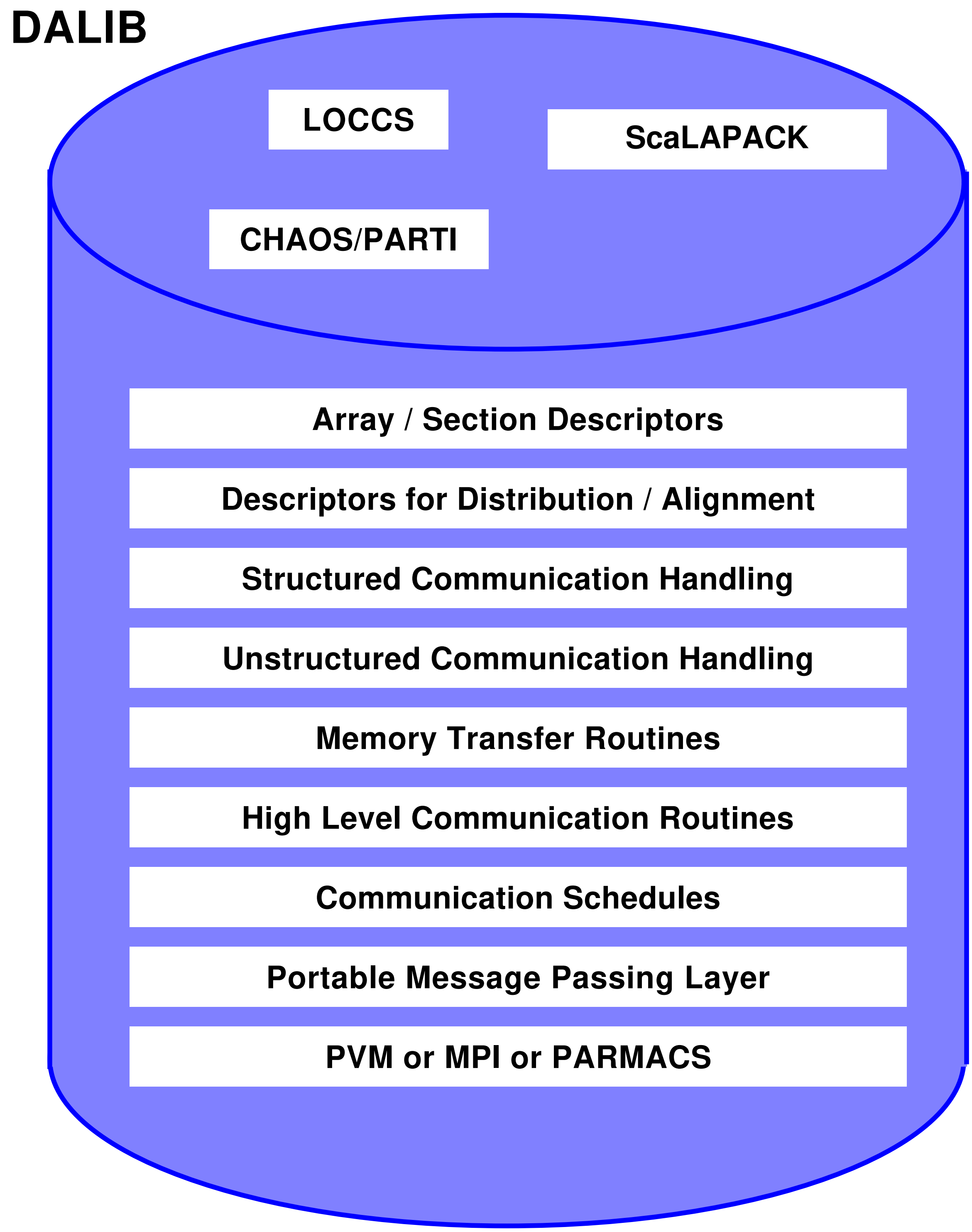
- no CYCLIC (n) distributions, n > 1

- dim argument of SUM, MINVAL, ... must be known
  at compile time

- MATMUL, DOTPRODUCT, DIAGONAL, PACK, UNPACK
  and RESHAPE are not available

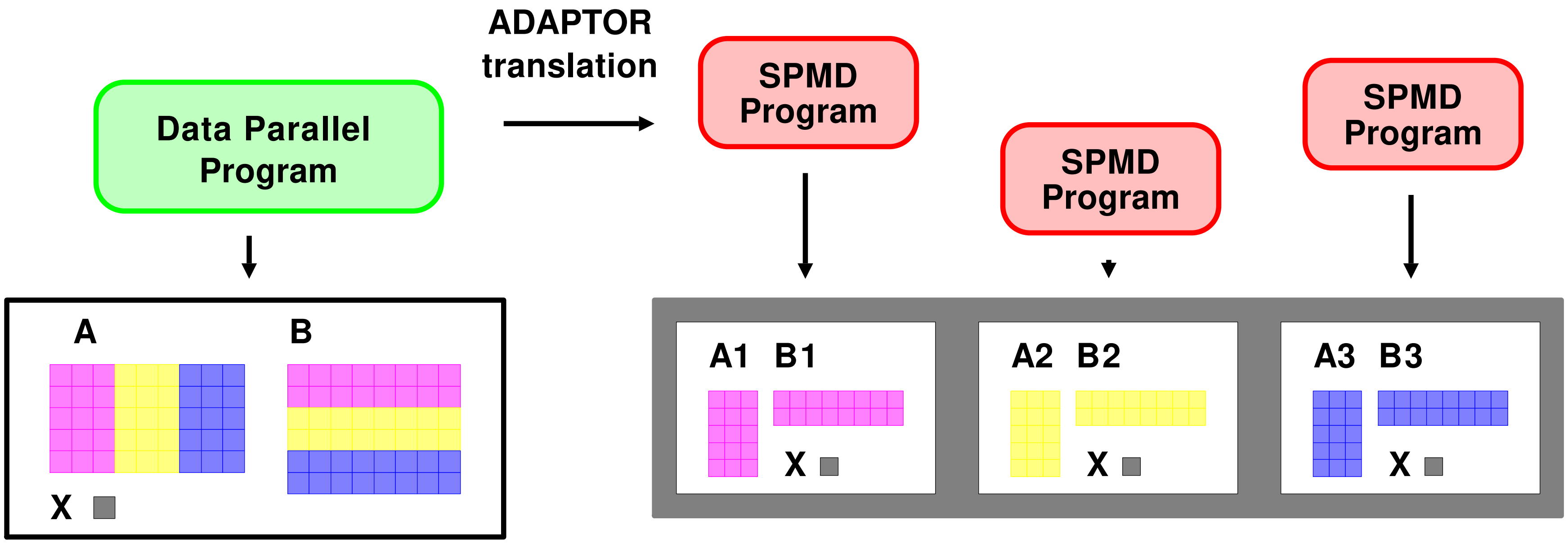
But:

- many HPF intrinsics (also the scatter operations)



LOCCS: Low Overhead Communication / Computation Subroutines

Functionality of HPF runtime system




```
         REAL A(5,9), B(6,8)
!HPF$ DISTRIBUTE A (*,BLOCK)
!HPF$ DISTRIBUTE B (BLOCK;*)
         REAL X
```

```
REAL, ALLOCATABLE :: A(:,:), B(:,:)
REAL X
call dalib_get_bounds (..., A_LOW, A_HIGH)
call dalib_get_bounds (..., B_LOW, B_HIGH)
...
ALLOCATE (A(1:5,A_LOW:A_HIGH))
ALLOCATE (B(B_LOW:B_HIGH, 1:8))
...
```

Automatic Partitioning within HPF compilers

Input source program (data parallel):

```
      REAL A (N), B
!HPF$ DISTRIBUTE A (BLOCK)
      A(10) = B     ! update of local data with replicated data
      B = A(10)     ! update of replicated data with local data
      A = 0.0
      A = A + 1.0
      B = sum (A)
```

Output source program (SPMD):

```
IF (dalib_is_local (A_DSP, 1, 10)) THEN A(10) = B

IF (dalib_is_local (A_DSP, 1, 10)) THEN B = A(10)

CALL dalib_array_broadcast (A_DSP, 1, 10, B, 4)

DO I1 = A_LOW1, A_HIGH1
   A(I1) = 0.0
END DO

DO I1 = A_LOW1, A_HIGH1
   A(I1) = A(I1) + 1.0
END DO

B = 0.0
DO I1 = A_LOW1, A_HIGH1
   B= B+ A(I1)
END DO
CALL dalib_reduction (B, 8)   ! 8 stands for sum of reals

...
```

```
      PARAMETER (N=100, NA=200, NB=50)
      REAL VAL(N), A(NA), B(NB)
      INTEGER IND1(N), IND2(N)
!HPF$ DISTRIBUTE (BLOCK) :: VAL, A, B
!HPF$ ALIGN (I) WITH VAL(I) :: IND1, IND2

      FORALL (I=1:N)
     &   A(IND1(I)) = B(IND2(I)) + VAL(I)
```

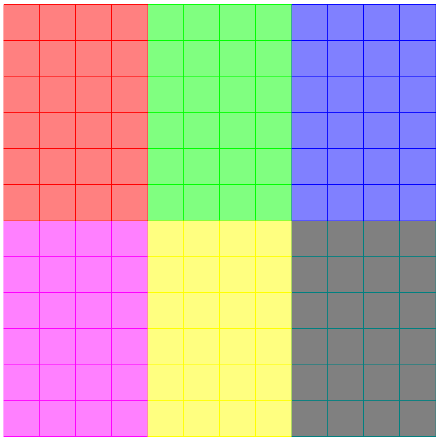
↓

```
      REAL TMP1(N), TMP2(N)
!HPF$ ALIGN(I) WITH VAL (I) :: TMP1, TMP2

      FORALL (I=1:N)                      ! pre-fetch
     &    TMP1(I) = B(IND2(I))

      FORALL (I=1:N)
     &    TMP2(I) = TMP1(I) + VAL(I)   ! local

      FORALL (I=1:N)
     &     A(IND1(I)) = TMP2(I)          ! post-store
```

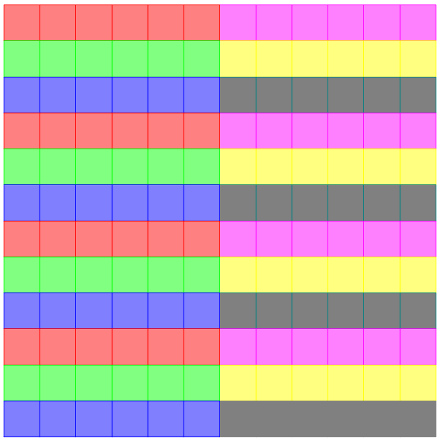
```
        REAL  A(N, N), B(N,N)
!HPF$ DISTRIBUTE A (BLOCK, BLOCK) ONTO P1
!HPF$ DISTRIBUTE B (CYCLIC, BLOCK) ONTO P2
```

- Processor 1
- Processor 2
- Processor 3
- Processor 4
- Processor 5
- Processor 6

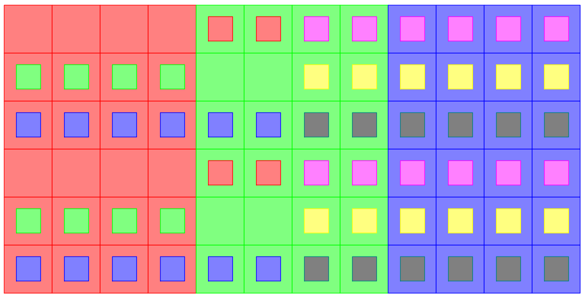
A



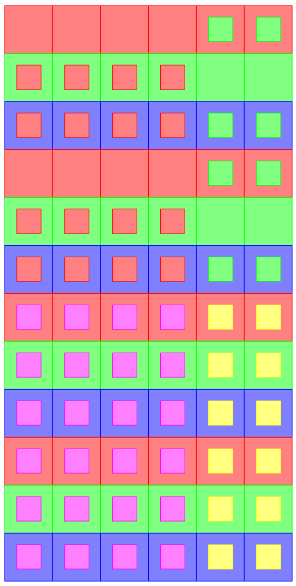
B



B = A



P1 - P3 : send



P1 - P3 : recv

Communication of Regular Sections

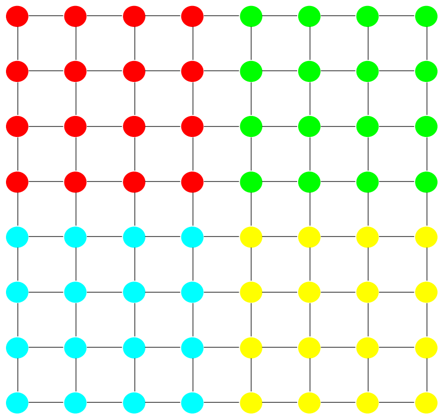
Optimisation Issue	Effect
Overlap Area	less local memory transfer less temporary data
Fusion Techniques - loop fusion - forward substitution - combining communic.	better cache usage less temporary data less communication less overhead
Statement Reordering	allows other optimizations
Reusing Comm. Pattern	reduces communication costs
Pipelining Techniques	using pipeline parallelism
Shared Memory Segments	reduces runtime overhead
Library Interface	using efficient libraries, e.g. ScaLAPACK of for parallel I/O

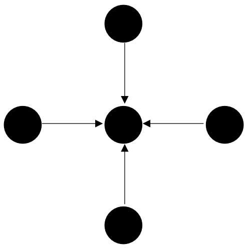
Not considered:

- overlapping computation / communication

Optimization Issues

```
        REAL A(N,N), B(N,N)
!HPF$ DISTRIBUTE (BLOCK, BLOCK) :: A
!HPF$ ALIGN B(I,J) WITH A(I,J)
        FORALL (I=2:N-1, J=2:N-1)
      &    A(I,J) = f ( B(I-1,J), B(I+1,J), B(I,J-1), B(I,J+1) )
```



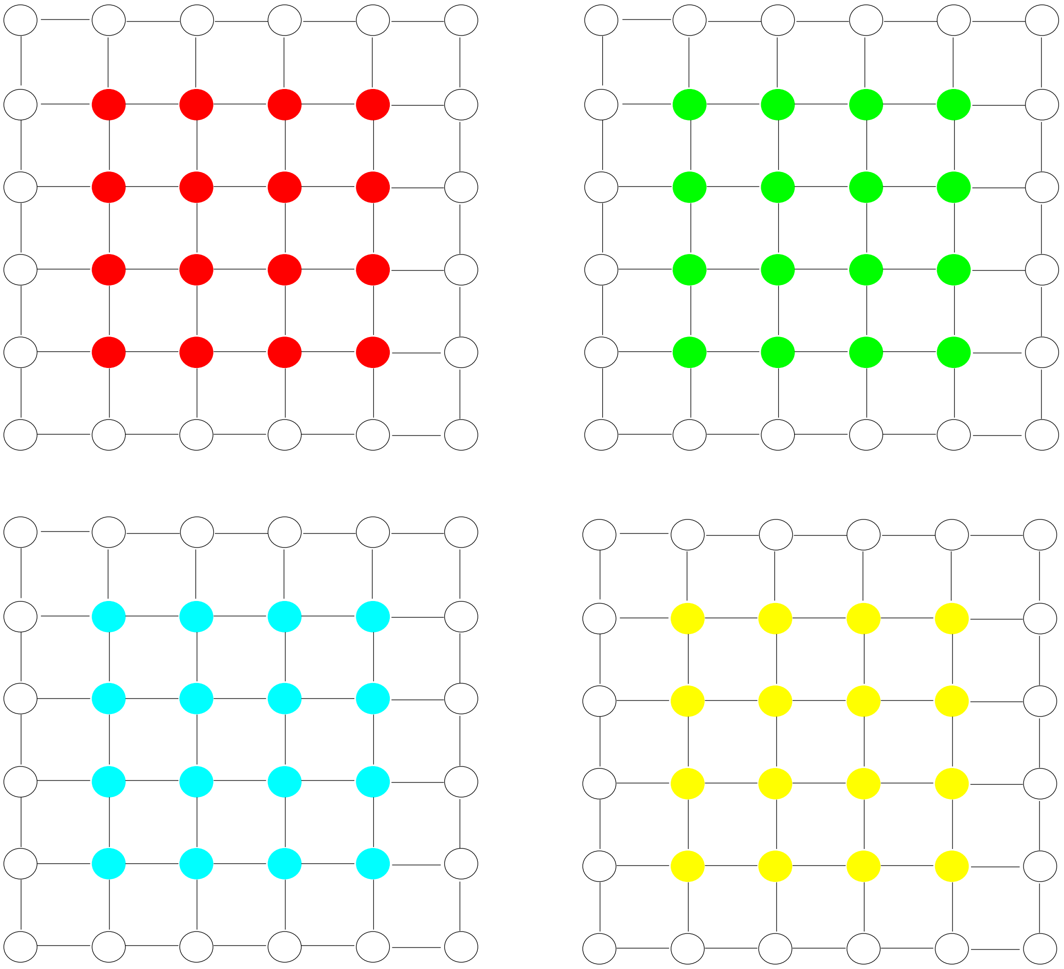


Stencil - Operation

```
  FORALL (I=2:N-1, J=2:N-1)
&    TMP1(I,J) = B(I-1, J)

  FORALL (I=2:N-1, J=2:N-1)
&    TMP2(I,J) = B(I+1, J)

  FORALL (I=2:N-1, J=2:N-1)
&    TMP3(I,J) = B(I, J-1)

  FORALL (I=2:N-1, J=2:N-1)
&    TMP4(I,J) = B(I, J+1)

  FORALL (I=2:N-1, J=2:N-1)
&    A(I,J) = f ( TMP1(I,J), TMP2(I,J), TMP3(I,J), TMP4(I,J) )
```

Need for Overlap Areas

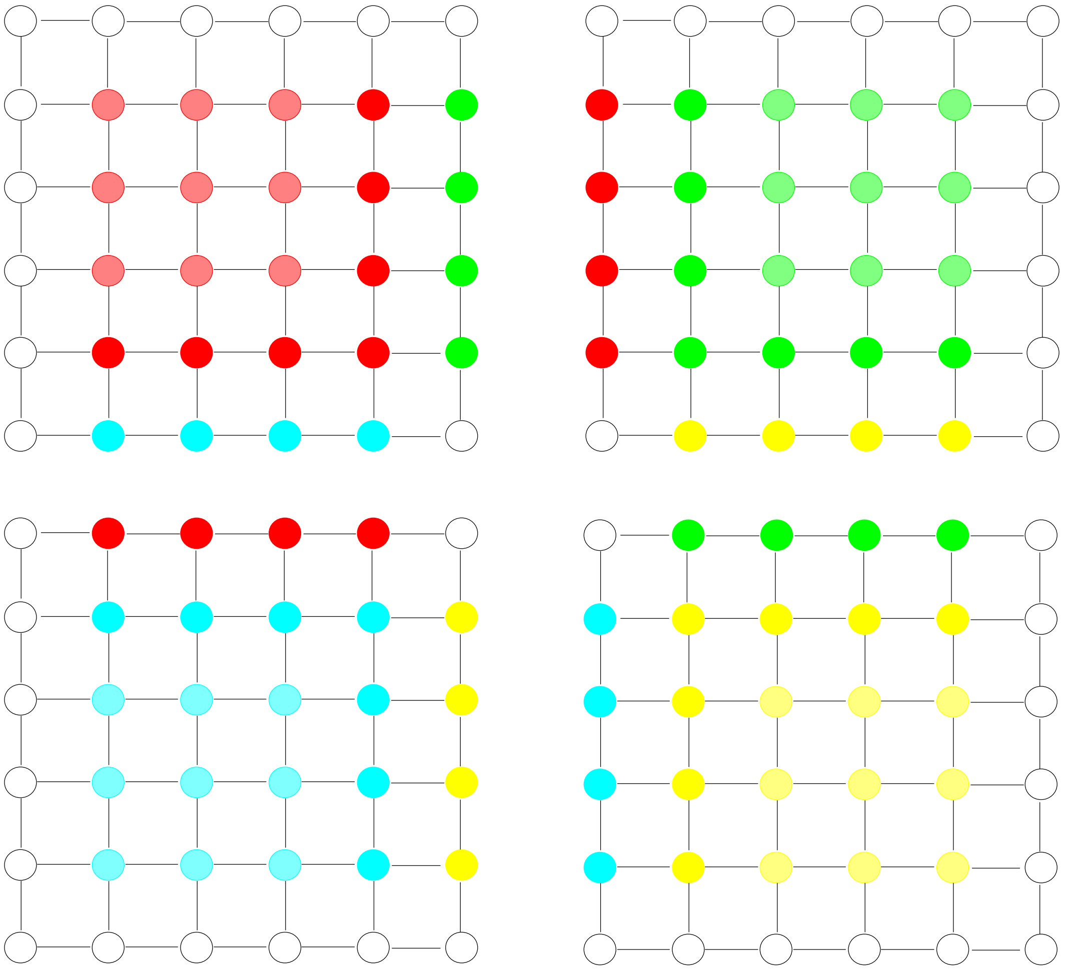
```
        REAL A(N,N), B(N,N)
!HPF$ DISTRIBUTE (BLOCK, BLOCK) :: A
!HPF$ ALIGN B(I,J) WITH A(I,J)
        FORALL (I=2:N-1, J=2:N-1)
     &    A(I,J) = f ( B(I-1,J), B(I+1,J), B(I,J-1), B(I,J+1) )
```



Overlap Area : every processor gets additional memory
for copies of non-local values

Use of Overlap Areas

```
       REAL A(N,N), B(N,N)
!HPF$ DISTRIBUTE (BLOCK, BLOCK) :: A
!HPF$ ALIGN B(I,J) WITH A(I,J)
```



```
   OVERLAP UPDATE B(2:N-1, 2:N-1) WITH [1:1,0:0]
   OVERLAP UPDATE B(2:N-1, 2:N-1) WITH [0:0,1:1]

      FORALL (I=2:N-1, J=2:N-1)
    &    A(I,J) = f ( B(I-1,J), B(I+1,J), B(I,J-1), B(I,J+1) )
```

Update of Overlap Areas

1. Compiler Support

- size of the overlap area ?

- overlap analysis determines the size of overlap areas
- but what happens with common arrays and dummy array

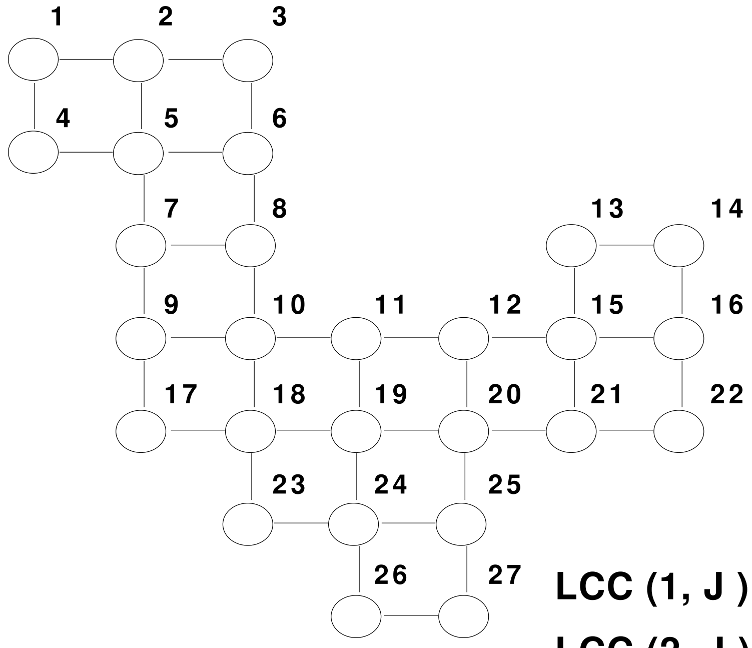
```
SUBROUTINE SUB (B, ...)
...
    A(I,J) = f (B(I,J-1), B(I-2,J) )
...
```

- update of the overlap area ?

```
B(I,J) = ....
...      = f (..., B(I-1, J+1), ...)
...      = g (..., B(I-1, J), B(I-1, J+1), ...)
```

2. Runtime Support

```
dalib_overlap_update (section_id, lov1, rov1, lov2, rov2, ...)

section_descriptor section_id;
int lov1, rov1, lov2, rov2, ... ;

/* efficient communication for exchanging boundary values */
```



real A (N), B(N)

integer LCC (4, N)

LCC (1, J) is south neighbor of cell J

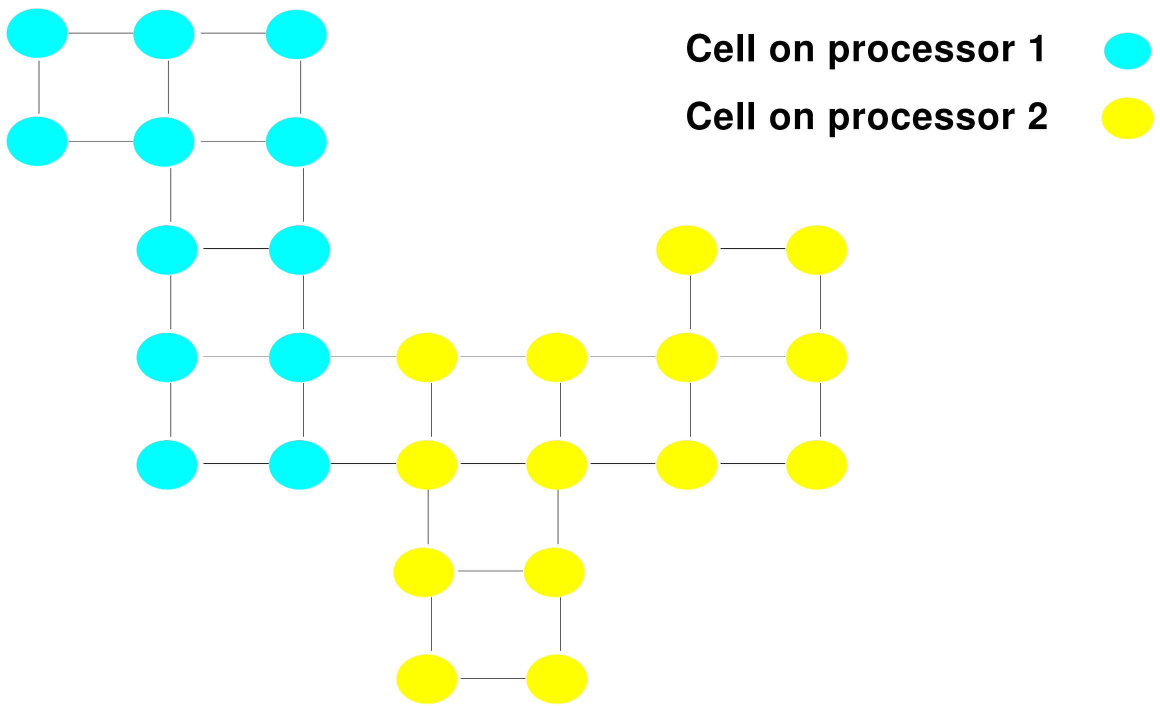
LCC (2, J) is east neighbor of cell J

LCC (3, J) is north neighbor of cell J

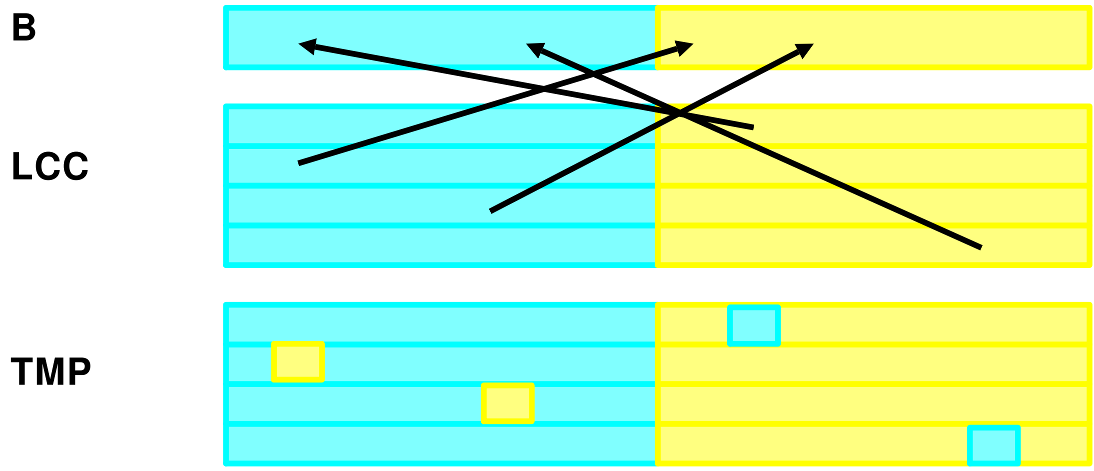
LCC (4, J) is west neighbor of cell J

```
        REAL A(N), B(N)
        INTEGER LCC (4, N)
!HPF$ DISTRIBUTE (BLOCK) :: A
!HPF$ ALIGN B(I) WITH A(I)
!HPF$ ALIGN LCC(*,I) WITH A(I)
        ....
          FORALL (I=1:N)
        &   A(I) = f ( B(LCC(1,I)), B(LCC(2,I)),
        &              B(LCC(3,I)), B(LCC(4,I)) )
```

Unstructured Grids / Indirect Addressing

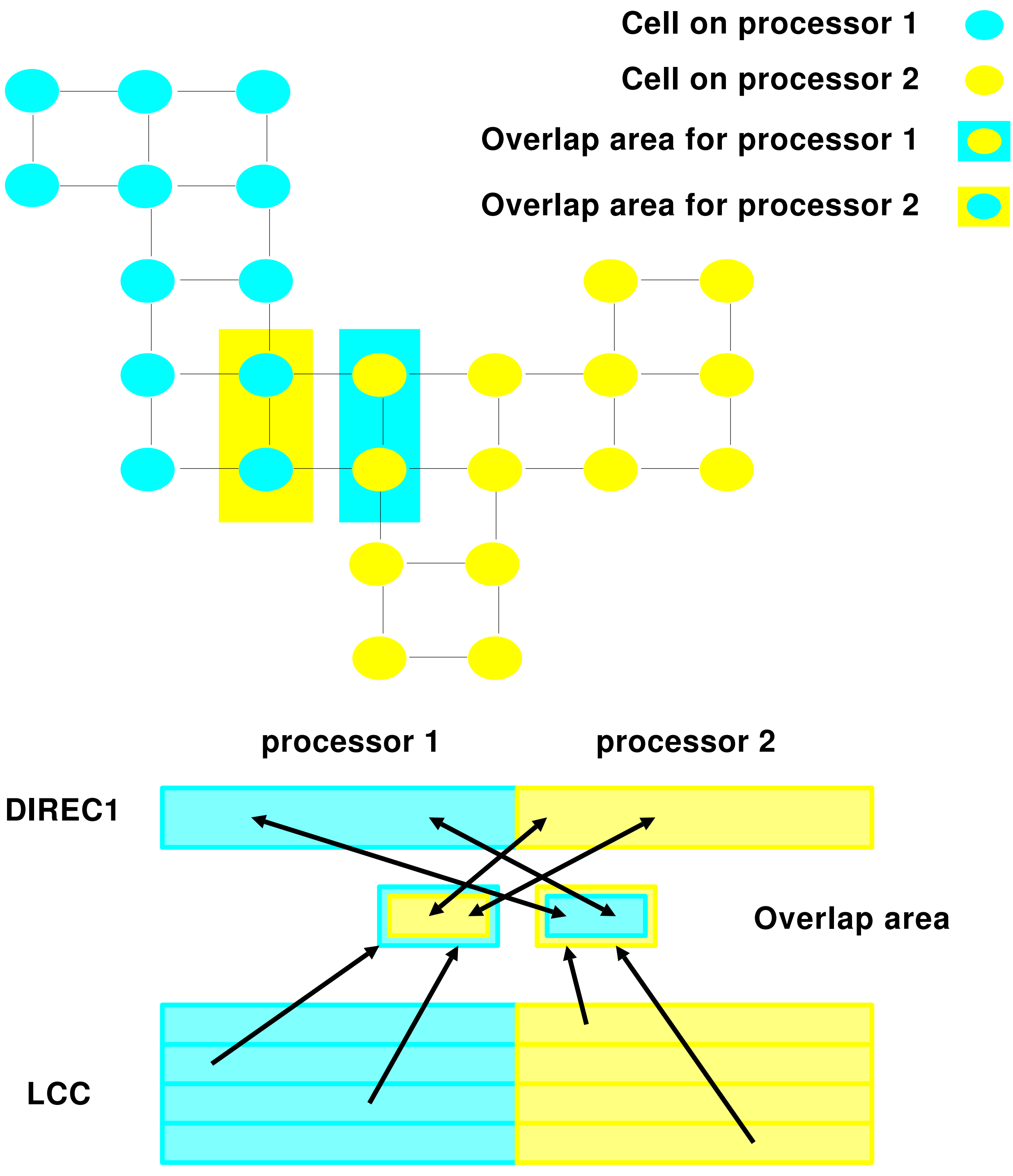



```
      REAL TMP (4, N)
!HPF$ ALIGN TMP(K,I) WITH LCC(K,I)

       FORALL ( I=1:N, K=1:4)        ! get the values of the neighbors
     &    TMP (K,I) = B ( LCC (K,I))
```



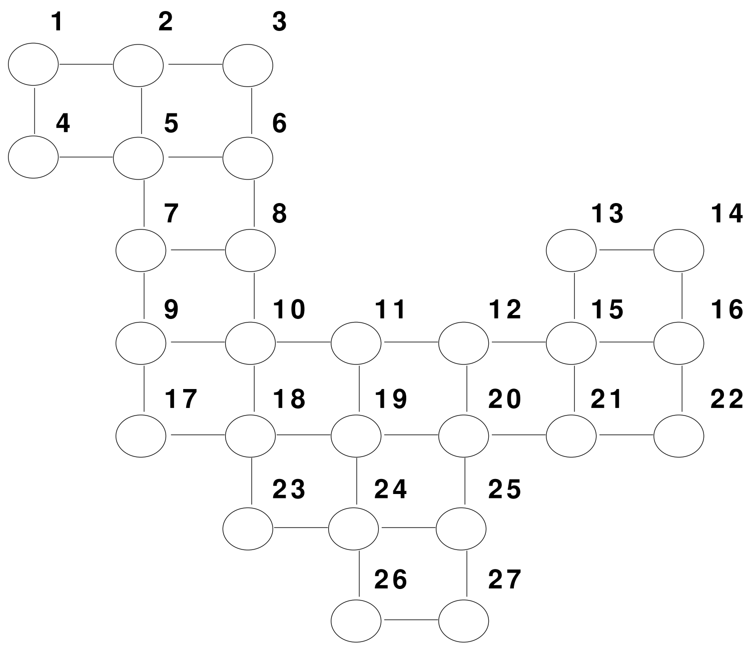
Most data movement is local copy !!

Data Movement for Indirect Addressing



Pointers to non-local data become pointers to overlap area, overlap area is replicated data from other processors

Overlap Areas for Indirect Addressing



real A (N), B(N)

integer LCC (4, N)

```
        REAL A(N), B(N)
        INTEGER LCC (4, N)
!HPF$ DISTRIBUTE (BLOCK) :: A
!HPF$ ALIGN B(I) WITH A(I)
!HPF$ ALIGN LCC(*,I) WITH A(I)
!ADP$ SHARED B                        ! make B to a shared array
        ....
         FORALL (I=1:N)
     &      B(I) = A(I)               ! copy local data into shared area
         FORALL (I=1:N)               ! compiler generates no comm.
        &   A(I) = f ( B(LCC(1,I)), B(LCC(2,I)),
        &              B(LCC(3,I)), B(LCC(4,I)) )
```

Using Shared Arrays

Example (from the AVL Fire GCCG benchmark solver):

```
        REAL ADXOR1(N), ADXOR2(N)
        REAL DIREC2(N)
!HPF$ ALIGN (I) WITH DIREC2 (I) :: ADXOR1( ADXOR2
        ...
        OC1 = sum (ADXOR1(1:N) * DIREC2 (1:N) )
        OC1 = OC1 / CNORM1
        OC2 = sum (ADXOR2(1:N) * DIREC2 (1:N) )
        OC2 = OC2 / CNORM2
```

```
        OC1 = 0.0
        DO I = DIREC2_LOW, DIREC2_HIGH
            OC1 = OC1 + ADXOR1(I) * DIREC2 (I)
        END DO
        GLOBAL REDUCE (sum, OC1)
        OC1 = OC1 / CNORM1
        OC2 = 0.0
        DO I = DIREC2_LOW, DIREC2_HIGH
            OC2 = OC2 + ADXOR2(I) * DIREC2 (I)
        END DO
        GLOBAL REDUCE (sum, OC2)
        OC2 = OC2 / CNORM2
```

```
        OC1 = 0.0                                     ! after fusion
        OC2 = 0.0
        DO I = DIREC2_LOW, DIREC2_HIGH
            OC1 = OC1 + ADXOR1(I) * DIREC2 (I)
            OC2 = OC2 + ADXOR2(I) * DIREC2 (I)
        END DO
        GLOBAL REDUCE2  (sum, OC1, sum, OC2)
        OC1 = OC1 / CNORM1
        OC2 = OC2 / CNORM2
```

Fusion Techniques

ScaLAPACK

```
CALL PDGEMM (TRANSA, TRANSB,
             M, N, K, ALPHA,
             A, IA, JA, DESC_A,
             B, IB, JB, DESC_B, BETA,
             C, IC, JC, DESC_C)
```

C = BETA * C + ALPHA * op(A) * op(B)

op(A) is M x K, op(B) is K x N, C is M x N

HPF Library Interface:

```
CALL HPF_DGEMM (TRANSA, TRANSB, ALPHA, A, B, BETA, C)
```

Solution 1 : use HPF EXTRINSIC

- portable
- use GLOBAL_xxx routines
- requires full HPF compiler

Solution 2 : write small C Interface

- very simple due to array descriptors
- not portable

Library Interface to ScaLAPACK

Machine: Meiko CS-2 (University of Vienna)

Problem size : 1024 x 1024

Distribution : (block, block)

ADAPTOR: version 3.0

F77: Fortran 3.0.1 from SUN

	1	2	4	8	16	32
not optimized		124.70 s	58.37 s	25.81 s	11.55 s	6.41 s
optimized	66.71 s	33.59 s	16.70 s	8.17 s	3.90 s	1.93 s
Speed-Up		3.71	3.50	3.16	2.96	3.32
MFlops	12.8	25.4	51.0	104.3	218.3	440.4

Results: Shallow Water Equation

HYDFLO

~ 1500 lines of code

three-dimensional compressible hydrodynamics code based on an explicit two-step Lax-Wendroff finite difference scheme
regular Eulerian mesh in a regular domain (3D, (*,block,block))
using stencil operations

Meiko CS-2	1	2	4
not optimized	12.32 s	5.74 s	2.93 s
optimized	8.49 s	4.30 s	2.27 s
Speed-Up	1.45	1.44	1.29

MOMA

~ 5500 lines of code

Free surface ocean model, based on the Bryan-Cox-Shutner code
D.J. Webb, Institute of Oceanographic Sciences, U.K.
CM Fortran version by Mike Ashworth, NERC

Meiko CS-2	1	2	4
not optimized	11.14 s	5.82 s	3.61 s
optimized	8.36 s	4.80 s	2.89 s
Speed-Up	1.33	1.21	1.25

First Results (on large codes)

AVL FIRE code

- interactive fluid dynamics package
- 3D, 2D unsteady and steady simulations
- complex geometries
- indirect addressing

Benchmark solver

- GCCG : conjugate gradients, orthomin with diagonal scaling
- about 400 lines of code
- main loop is iteration over the geometry using indirect addressing

a) Results on Intel Paragon, medium data set

47312 cells	1	2	4	8	16
HPF version	680 s	350 s	177 s	89 s	46 s
+ reuse schedule	205s	103 s	53 s	27 s	15 s
+ overlap areas	137 s	70 s	37 s	20 s	11 s
HPF, shared array	148 s	84 s	51 s	37 s	35 s
MPI version	100 s	-	22 s	13 s	8 s

b) Results on IBM SP 2, big data set

108 000 cells	1	2	4	8	16
serial code	73.9 s				
opt. HPF version	79.4 s	42.4 s	22.4 s	12.5 s	8.2 s

Results: AVL Fire Code, GCCG Benchmark

Conclusions:

HPF is

- very expressive and powerful
- feasible for many scientific applications
- the future of Connection Machine Fortran
- competitive

HPF compilers

- are already available and will improve
- can still use a lot of optimization techniques

For the efficient use of HPF users

- still have to think about parallelism and data locality
- have to understand what the compilers do
- should claim for more parallel libraries