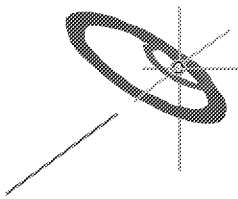
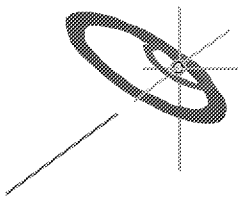
# Data-Parallelism for Engineering-Applications: HPF and HPF-2

Alfred Geiger
High-Performance Computing-Center Stuttgart (HLRS)
geiger@hlrs.de

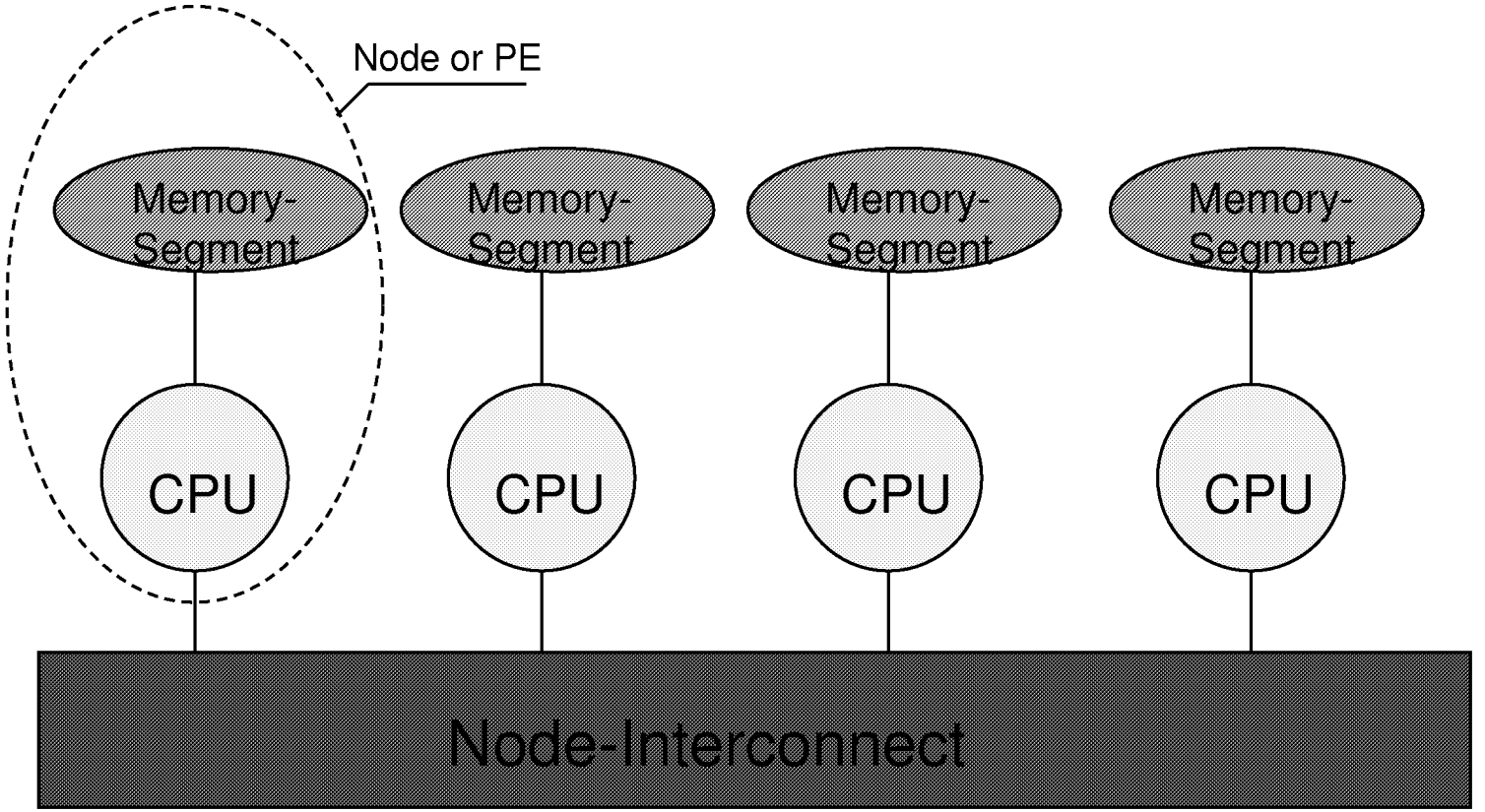


## Data-Parallelism for Engineering-Applications: HPF and HPF-2

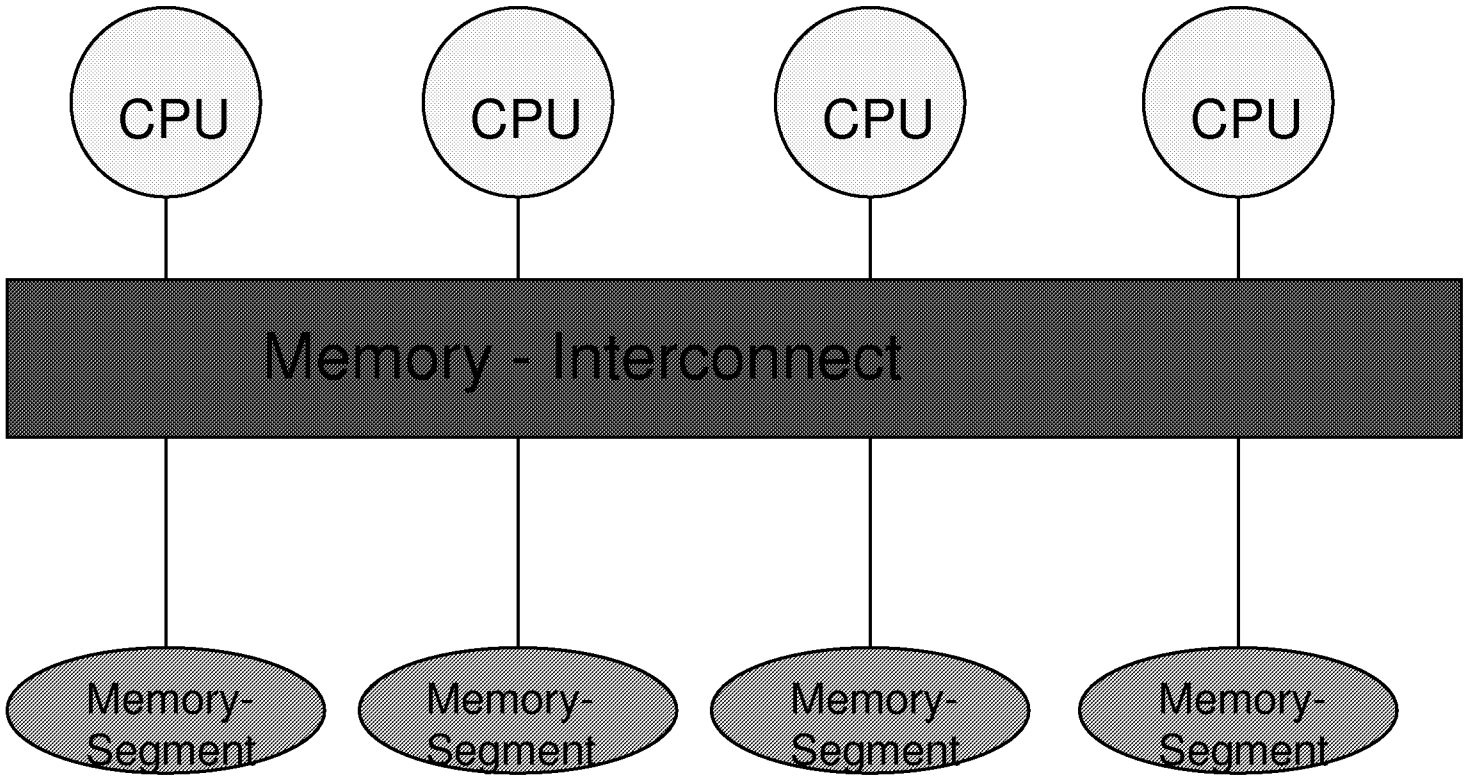
- Programming-Models and Data-Parallelism
- HPF-Basics
- Data-Mapping
- Communication
- Expression of Parallelism
- HPF-2
- Application-Scenarios and Limits



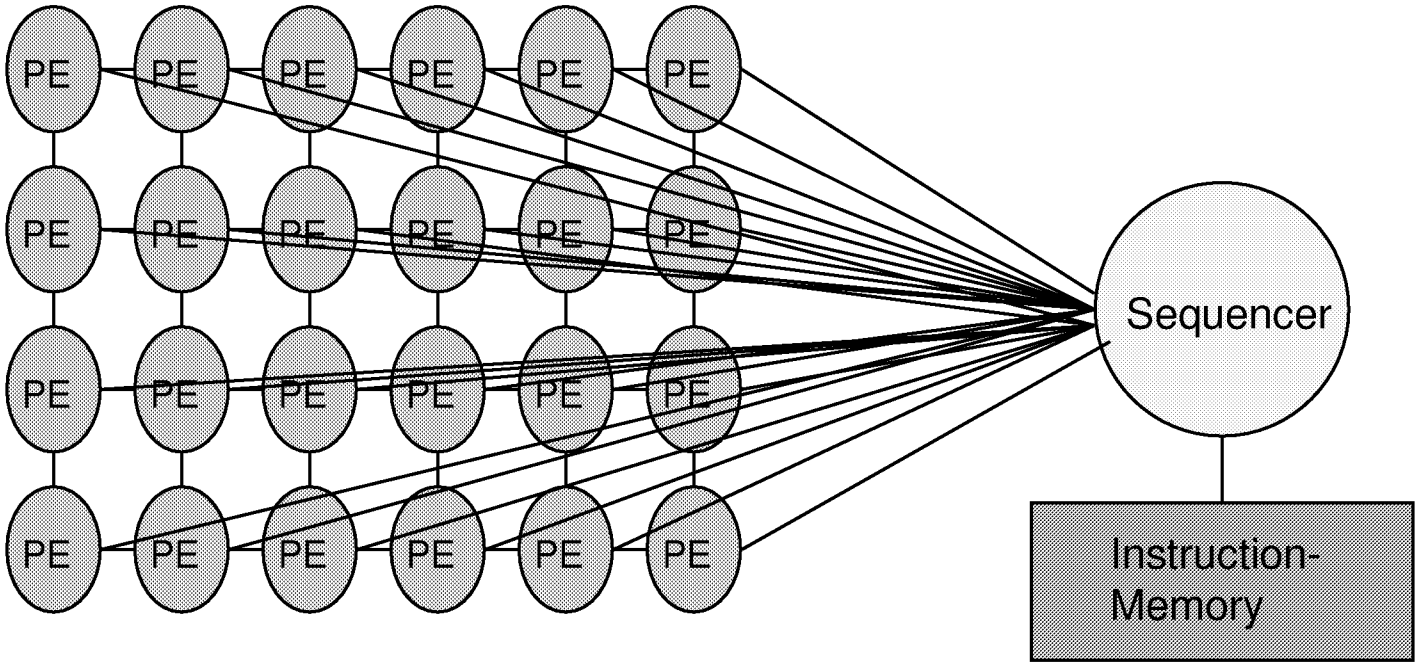
# Multicomputer



## Multiprocessor



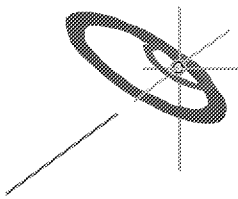
# Array - Processor




PE : CPU + Data - Memory

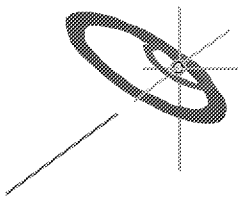
# Parallelism and Hardware

| | Pipelining | Superscalar | Array | Multiprocessor | Multicomputer |
|---|---|---|---|---|---|
| Explicit | + | + | + | + | + |
| Implicit Job | - | - | - | + | + |
| Implicit Task | - | - | - | + | + |
| Implicit Statement | - | - | - | + | + |
| Implicit Instruction | - | + | - | - | - |

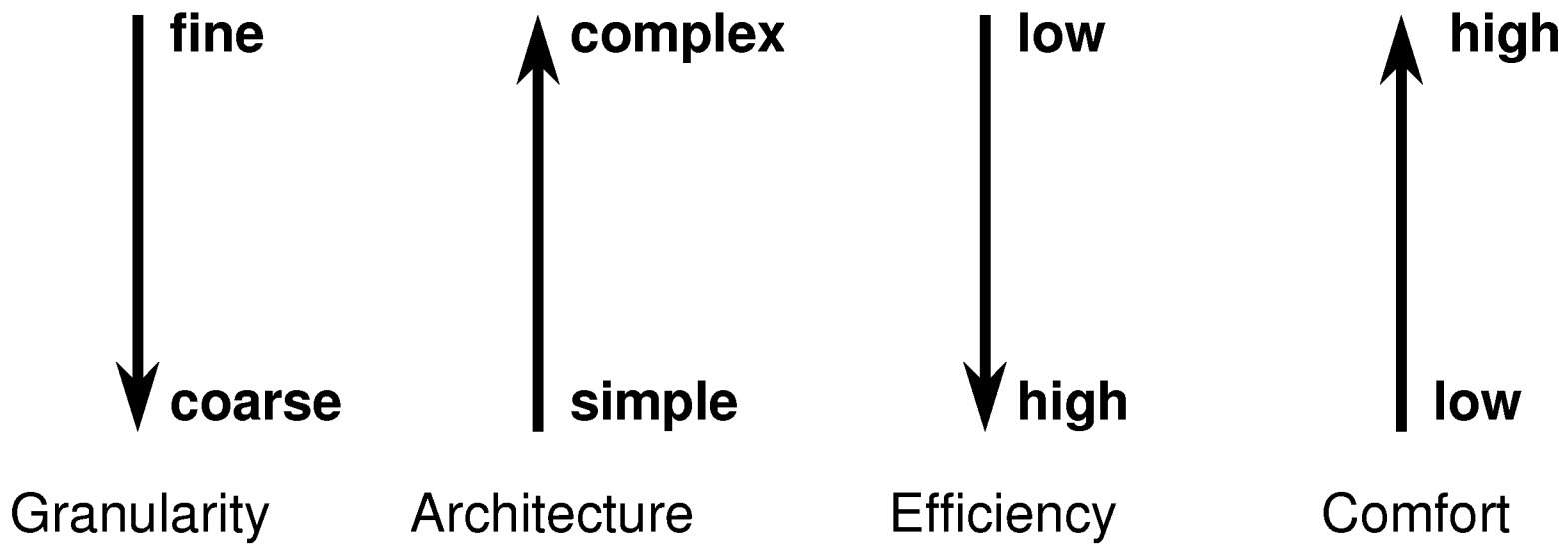


## Programming-Models for NUMA

- Message-Passing
  - Explicit Data-Distribution / Explicit Communication
  - The Standard: MPI
- Data-/Worksharing
  - Explicit Data-Distribution / Implicit Communication
  - The Standard: HPF
- Distributed Shared-Memory
  - Implicit Data-Distribution / Implicit Communication
  - Actually no Standard

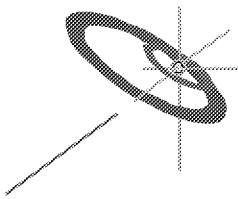


## Granularity, Architecture and Comfort



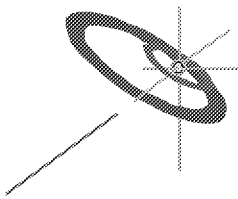
## HPF - History

- Defined by the High Performance Fortran Forum (HPFF)
- Proposed at Supercomputing'91
- Final Draft in June 1993
- Definition of HPF-2 in 1994 and 1995
- Influences from CM-Fortran, MasPar, DEC, Fortran-D, Vienna Fortran
- Goal: Portable Language for all platforms



## HPF-Basics 1

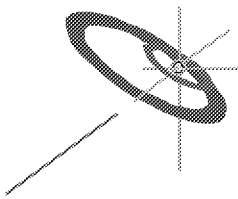
- Parallel Extension to FORTRAN-90
  - Input for further standardisation of FORTRAN
  - No conflict and minimal deviation from existing standards
- Mapping of data-objects (arrays) and instructions to parallel architectures
- Directives
- Shared-Memory like Parallelism
  - Loop and Block-Level
  - Local changes
  - Ease of Porting
  - Incremental Parallelisation and Optimisation
- Global Data
  - Restricted usability together with Message-Passing



## HPF-Basics 2

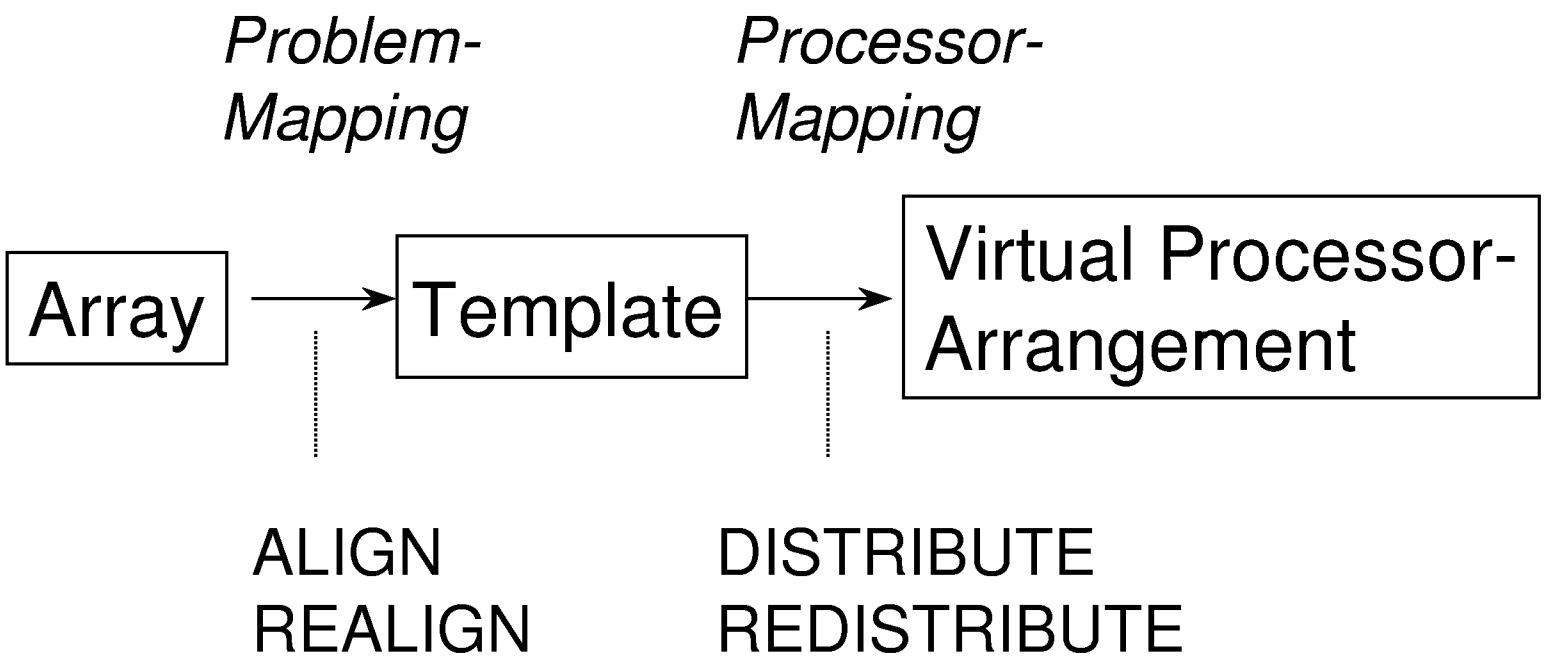
- Global Name-Space
  - Distributed arrays may not be used when storage- or sequence-association is needed (e.g. collapsing dimensions).

    REAL A(1:N,1:N)
    global address-space: A(N+1) = A(1,2)
    global name-space: A(N+1) >< A(1,2)
- Worksharing according to the Set-Compute-Rule
  - The node which has the data in the LHS of a statement does the work
  - Influencable with ON clause
  - General worksharing with LOCAL subroutines
- Single or multiple threads of control (implementation dependent)
  - Less synchronisation in multithreaded case
  - Replication of Scalars in multithreaded case



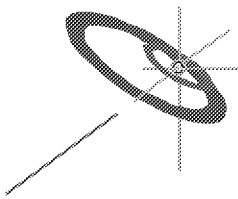
## Concept of Data-Mapping

# Two-step Mapping-Concept:



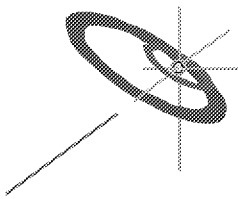
# Goals of Data-Mapping

- Load-Balancing: Same load on all nodes
- Locality: Data used together on same node
- Minimal Contention: Data used in parallel on different processors
- Potential Conflicts:
  - Maximal Locality: All data on same processor
  - Minimal Contention: All data on different processors



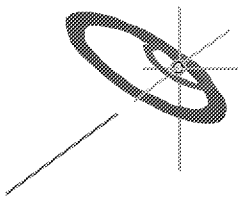
## Problem-Mapping: TEMPLATE, ALIGN and REALIGN

- **!HPF$ TEMPLATE t(1:n,1:n)**
  - Abstract space of indexed positions
  - Declaration
- **!HPF$ ALIGN a WITH t**
  - Mapping of arrays relative to templates or other arrays
  - Declaration
  - e.g. transposed CHPF$ ALIGN a(j,i) WITH t(i,j)
  - e.g. multigrid CHPF$ ALIGN c(i,j) WITH f(2*i,2*j)
- **!HPF$ REALIGN a(j,i) WITH t(i,j)**
  - Realignment during Runtime
  - Executable Statement
  - Arrays must have DYNAMIC attribute (f90)



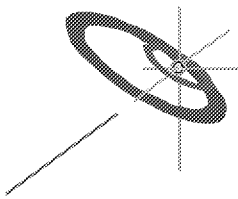
# Processor-Mapping: PROCESSORS, DISTRIBUTE and REDISTRIBUTE

- **!HPF$ PROCESSORS p(1:n,1:n)**
  - Virtual processor-topology
  - Only grid-topologies possible (syntax of FORTRAN-arrays)
  - Declaration
- **!HPF$ DISTRIBUTE t(BLOCK,CYCLIC) ONTO p**
  - Mapping of templates to processor arrays
    - BLOCK
    - CYCLIC
    - CYCLIC(n)
    - BLOCK(n)
  - Declaration
- **!HPF$ REDISTRIBUTE t ONTO p2**
  - Redistribution during runtime
  - Executable Statement
  - Arrays must have DYNAMIC attribute (f90)



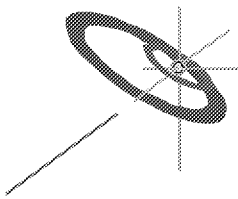
## Communication-Rules

- Communication is implicit and hidden from the user.
- Communication takes place (default) when operands are on different processors (indicated by different colors)
  - a(i) = b(j) + b(j+1)  Comunicate b(j), b(j+1)
  - x = a(i) + b(j)  Communicate {a(i), b(j)}
- By default operations are performed on the PE which owns the LHS.
- In implementations following the SPMD-model, scalars are replicated and synchronized.



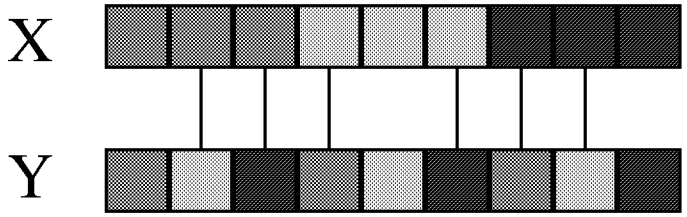
## Minimising Communication

- BLOCK-Distributions are optimal for algorithms with nearest-neighbour type locality. Problems can occur from arrays of different size.
- CYCLIC-Distributions have optimal load-balancing characteristics but rarely good communication-bahaviour.
- Strides produce high communication-overhead.
- Broadcast (e.g. transpose) is the worst case.
- Dynamic change of distributions is expensive.



## Communication-Requirements

Communicated array-elements (Data-Volume):
All elements that are combined residing on different nodes
Latencies:
Data-Volume (sequential case)
Number of distinct ordered pairs of PEs (packed case)
1 (parallel case)

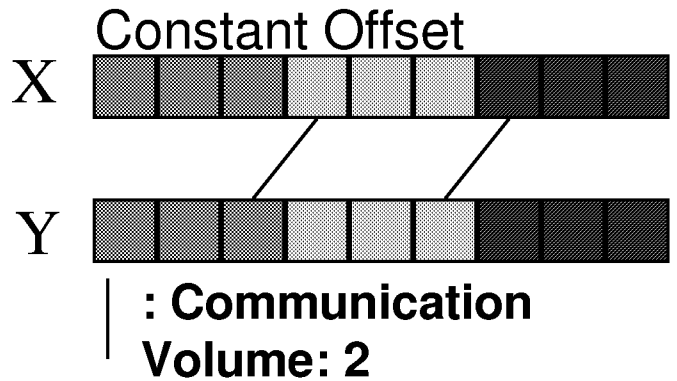



| : Communication
Volume: 6
Latencies: 6

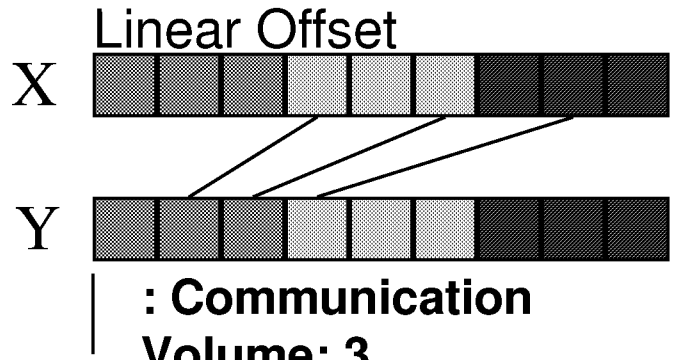
Example:

```
        REAL X(1:9), Y(1:9)
!HPF$ DISTRIBUTE X(BLOCK)
!HPF$ DISTRIBUTE Y(CYCLIC)
        DO I = 1,9,1
                Y(I) = X(I)
        END DO
```

## Communication: BLOCK



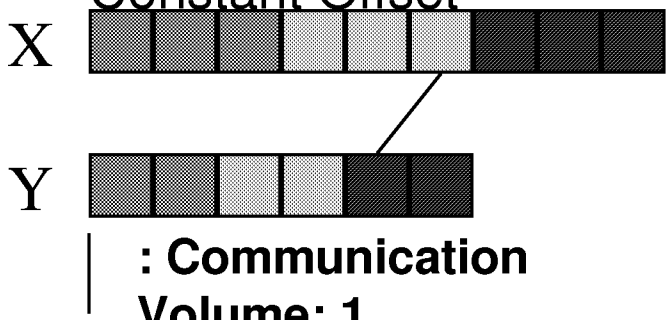

```
      REAL X(1:9), Y(1:9)
!HPF$ DISTRIBUTE X(BLOCK)
!HPF$ DISTRIBUTE Y(BLOCK)
      DO I = 1,8,1
            Y(I) = X(I+1)
      END DO
```




```
      REAL X(1:9), Y(1:9)
!HPF$ DISTRIBUTE X(BLOCK)
!HPF$ DISTRIBUTE Y(BLOCK)
      DO I = 1,4,1
            Y(I) = X(2*I)
      END DO
```

## Communication: BLOCK, Non-Matching Dimensions

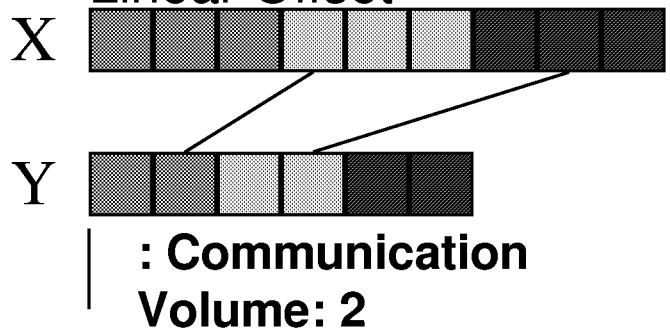
Constant Offset




**: Communication**
**Volume: 1**
**Latencies: 1**

```
      REAL X(1:9), Y(1:6)
!HPF$ DISTRIBUTE X(BLOCK)
!HPF$ DISTRIBUTE Y(BLOCK)
      DO I = 1,6,1
            Y(I) = X(I+1)
      END DO
```

Linear Offset

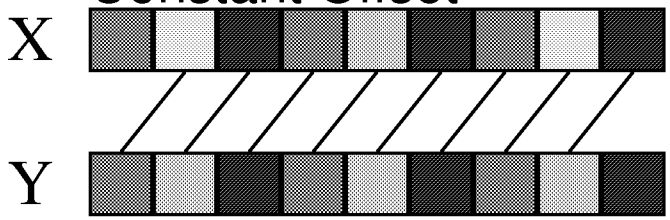



**: Communication**
**Volume: 2**
**Latencies: 2**

```
      REAL X(1:9), Y(1:6)
!HPF$ DISTRIBUTE X(BLOCK)
!HPF$ DISTRIBUTE Y(BLOCK)
      DO I = 1,4,1
            Y(I) = X(2*I)
      END DO
```

## Communication: CYCLIC

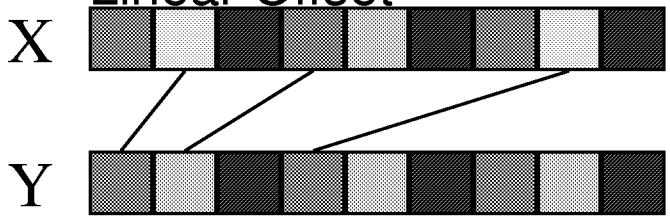
Constant Offset




**: Communication**
**Volume: 8**
**Latencies: 5(3)**

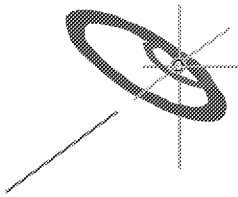
```
      REAL X(1:9), Y(1:9)
!HPF$ DISTRIBUTE X(CYCLIC)
!HPF$ DISTRIBUTE Y(CYCLIC)
      DO I = 1,8,1
            Y(I) = X(I+1)
      END DO
```

Linear Offset

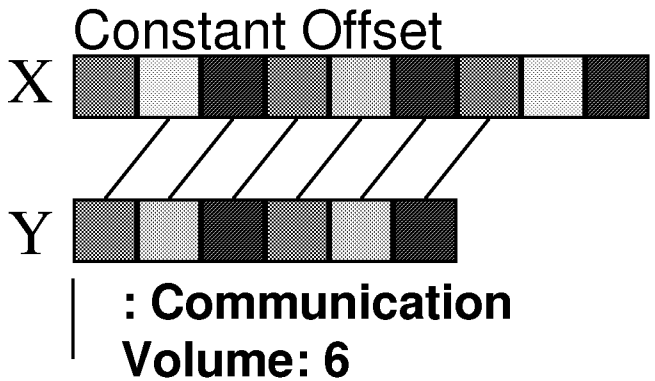



**: Communication**
**Volume: 3**
**Latencies: 3(2)**

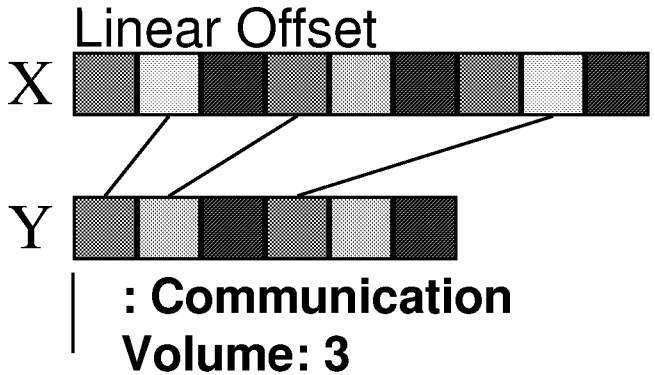
```
      REAL X(1:9), Y(1:9)
!HPF$ DISTRIBUTE X(CYCLIC)
!HPF$ DISTRIBUTE Y(CYCLIC)
      DO I = 1,4,1
            Y(I) = X(2*I)
      END DO
```



## Communication: CYCLIC, Non-Matching Dimensions



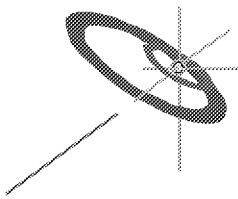

```
      REAL X(1:9), Y(1:6)
!HPF$ DISTRIBUTE X(CYCLIC)
!HPF$ DISTRIBUTE Y(CYCLIC)
      DO I = 1,6,1
            Y(I) = X(I+1)
      END DO
```




```
      REAL X(1:9), Y(1:6)
!HPF$ DISTRIBUTE X(CYCLIC)
!HPF$ DISTRIBUTE Y(CYCLIC)
      DO I = 1,4,1
            Y(I) = X(2*I)
      END DO
```

## Communication: Remarks

- Different Distributions:
  - Almost all elements are communicated
  - Global communication-pattern
- Indirect Distributions:
  - Can't be handled by HPF-1 (Feature in HPF-2)
  - Workaround by using preprocessors and intrinsics
- Different Array-Size:
  - Cyclic may help depending on access-pattern



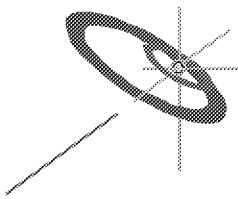
## Expression of Parallelism: FORALL

- Simultaneous assignment to a group of array-elements
- Generalized Loop
- Example

```
FORALL (i=1:n, j=1:m, y(i,j) .ne. 0.0)
            x(i,j) = 1.0/y(i,j)
END FORALL
```

≡≡≡

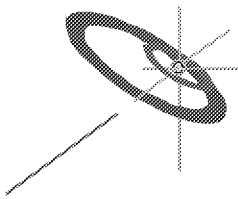
```
DO i = 1,n
        DO j = 1,m
                IF (y(i,j) .ne. 0.0) THEN
                        x(i,j) = 1.0/y(i,j)
                END IF
        END DO
END DO
```



## Expression of Parallelism: INDEPENDENT

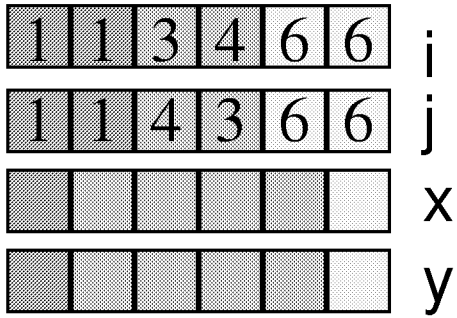
- Assertion to the compiler, that there are no dependencies in the following DO-loop or FORALL

```
!HPF$       INDEPENDENT(i)
            DO i = 1,100,1
                ......
            END DO
```



## Expression of Parallelism: Execute ON HOME

Suggest where iterations are performed by influencing the set-compute-rule



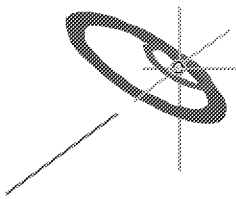

```
!HPF$ INDEPENDENT
      DO K = 1,6,1
!HPF$       ON HOME (I(K))
            Y(K) = X(I(K)) + X(J(K))
      END DO
```

Standard

```
Y(1) = X(1) + X(1)

Y(2) = X(1) + X(1)
Y(3) = X(3) + X(4)
Y(4) = X(4) + X(3)
Y(5) = X(6) + X(6)

Y(6) = X(6) + X(6)
```

ON HOME

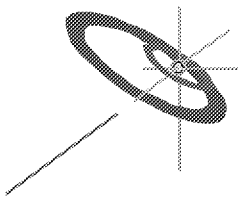
```
Y(1) = X(1) + X(1)
Y(2) = X(1) + X(1)

Y(3) = X(3) + X(4)
Y(4) = X(4) + X(3)

Y(5) = X(6) + X(6)
Y(6) = X(6) + X(6)
```



## Expression of Parallelism: PURE Procedures

- No change of global data which is not in parameter-list
- No change of global pointer-associations and data-mappings
- Recommendation: Make all procedures PURE in the first step of parallelisation with HPF
- Call by reference
- Example:

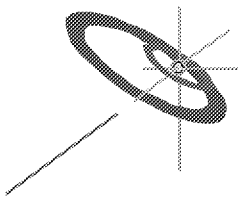
```
        FUNCTION f(x)
!HPF$   PURE f
        ......
        ......
        END FUNCTION
```



## Expression of Parallelism: LOCAL Procedures

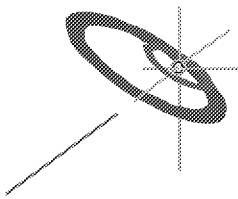
- Subroutines containing only local data
- Local address- and name-space
- Message-Passing between LOCAL procedures is possible
- Worksharing
- Call by value!!!
- Example:

```
                FUNCTION loc(xx,yy)
!HPF$           LOCAL
                .......
                .......
                END FUNCTION
```



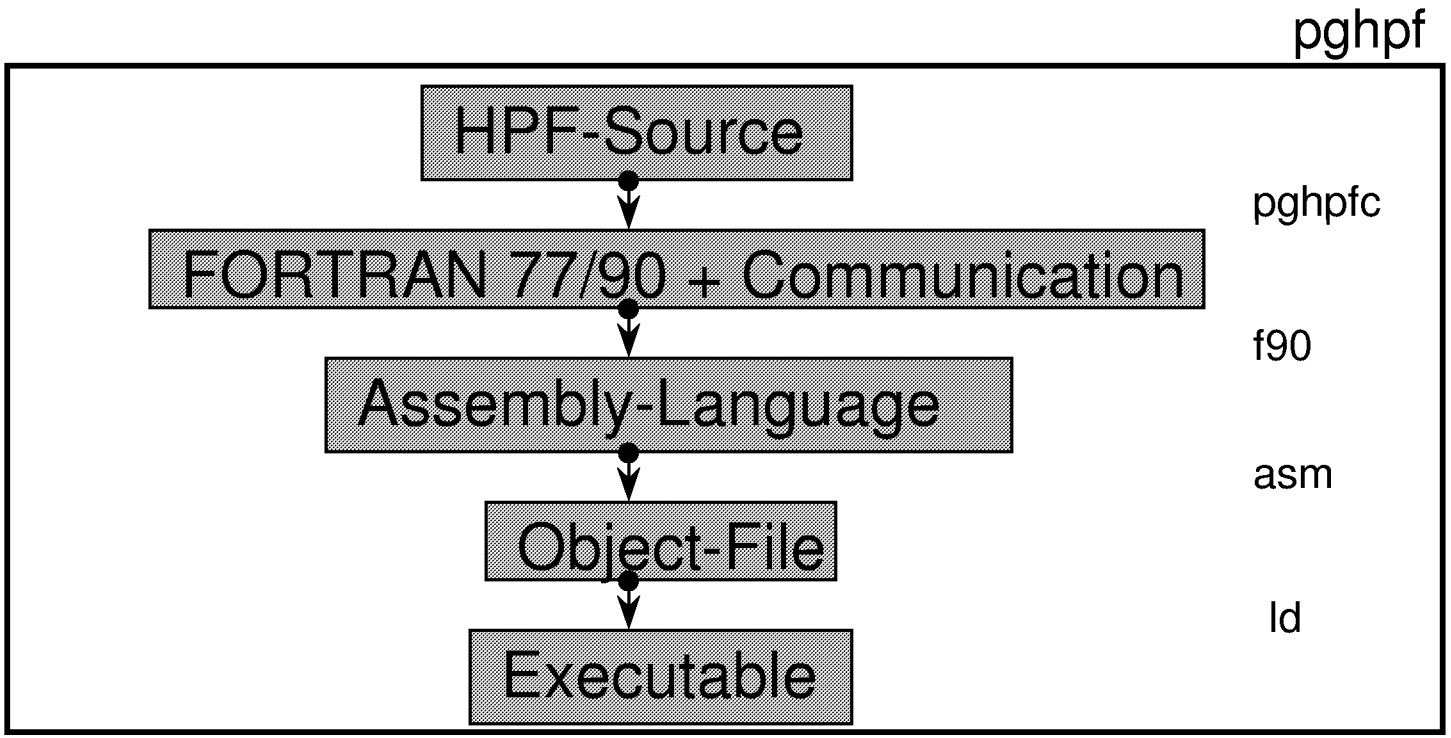
## Libraries and Intrinsics

- Parallel versions of FORTRAN-90 intrinsics
  - Arithmetics
  - Shift (e.g. CSHIFT)
  - Basic linear algebra (e.g. DOT_PRODUCT)
  - Global Operations (e.g. MAXVAL)
- HPF Intrinsics
  - Prefix, Suffix operations (e.g. ANY_PREFIX)
    Linear array functions in which the actual state depends on all previous/following array-elements
  - Gather/Scatter (e.g. MAXVAL_SCATTER)
  - Parallel-Runtime (e.g. NUMBER_OF_PROCESSORS)
  - Global Operations



## The PGI HPF-Compilation System

- Developped by Portland Group
- Available on Cray T3E, intel Paragon, IBM SP/2 and all Workstation/PC-Clusters

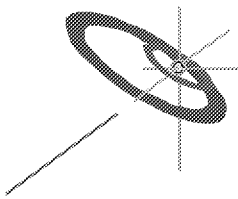


## PGHPF Compiler-Syntax

pghpf -Mhpfoption -f90option prog.hpf

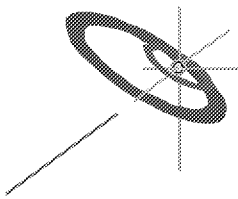
Important Options:

| | |
|---|---|
| -Mautopar | Automatic Parallelisation of Loops |
| -Mg | Debug-Option |
| -Minline | Inlineing of independent Loops |
| -Msequence | Create all variables in sequence |



## hpfsx - Compilation and Execution

- hpfsx is a preprocessor for f77sx
- Calling-Sequence : **hpfsx [hpf-options] -f77sx '[f77-options] <hpfprog.hpf>**
- Example: **hpfsx -O3 -R4 -f77sx '-o bscgal' bscgal.hpf**
- Diagnostic-Messages in hpfprog.diag
- Separate Compilation and Linking: Options!
- Execution: **mpisx <run-module> [options]**
- Mapping to Posix-Threads or UNIX-Processes



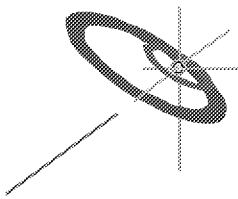
## hpfsx - Restrictions/Enhancements

- Based on f77sx instead of f90sx (until 8.1)
- Only f90 features: Array-descriptors :, * and INTENT, INTERFACE **but** no array-syntax (subarrays!)
- No complete Subset-HPF
- Enhancements start with **!HPX$**

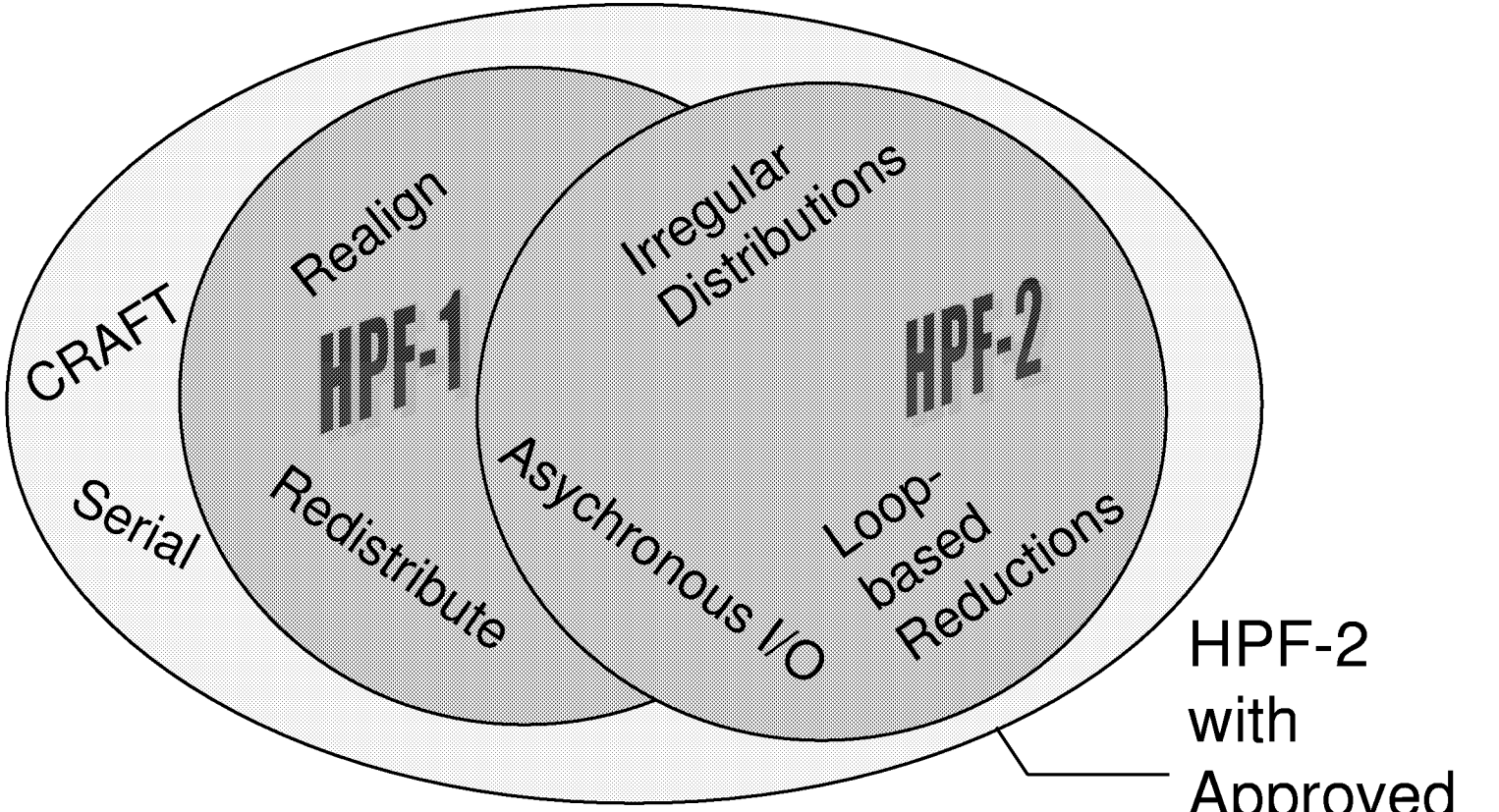
Mapping of Instructions (**ITEMPLATE, ITERATIONS**)

Threading (**SINGLE, OWNER, ALL**)

Control of Data-Communications (**NO_PRE_MOVE, NO_POST_MOVE**)

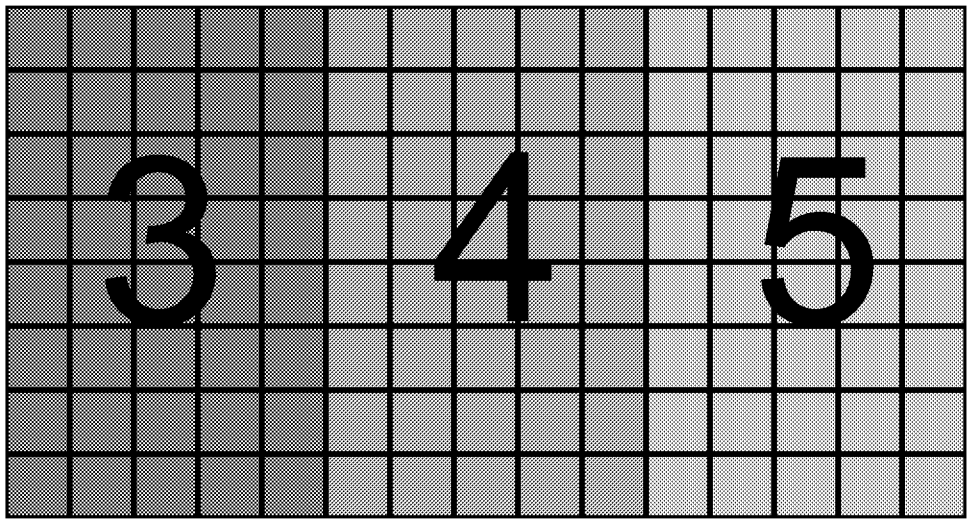


# HPF-2 and Approved Extensions



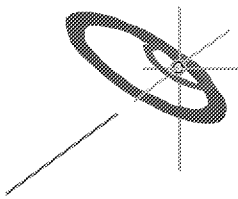
## HPF-2: Distribution to Processor-Subsets

```
!HPF$       PROCESSORS P(1:5)
!HPF$       DISTRIBUTE A(*,BLOCK) ONTO P(3:5)
```



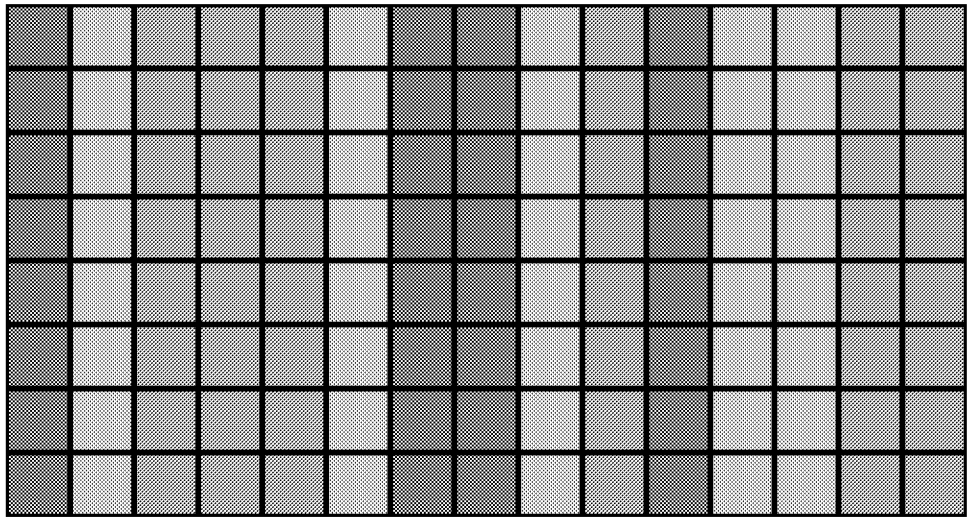
## HPF-2: Distribution of Derived Type Components

```
TYPE grid_variables
        REAL rho(1:50,1:100,1:50), p(1:50,1:100,1:50)
        !HPF$ DISTRIBUTE (BLOCK, BLOCK, *) :: rho, p
END TYPE
```



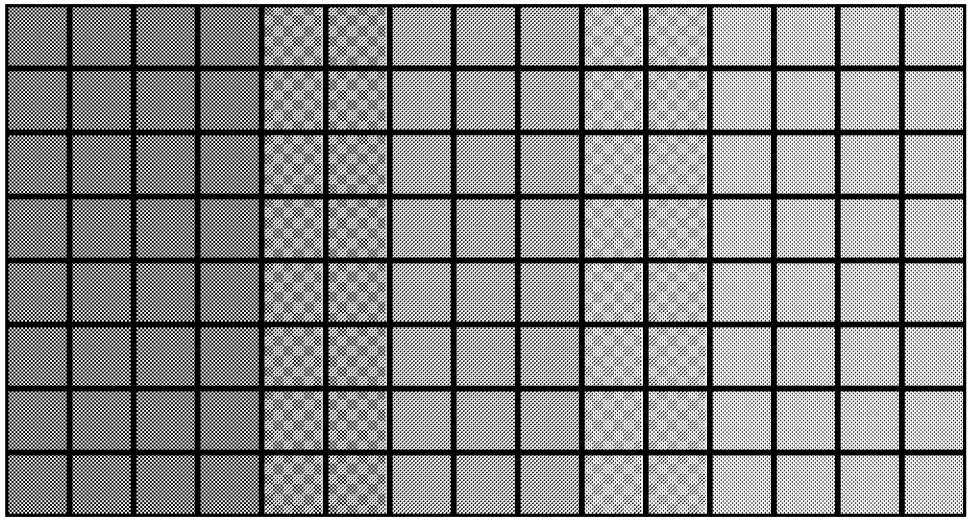
# HPF-2: Irregular Distributions

```
      INTEGER  map(1:15) = (/1,3,2,2,2,3,1,1,3,2,1,3,3,2,2/)
!HPF$ DISTRIBUTE A(*, INDIRECT(map))
```



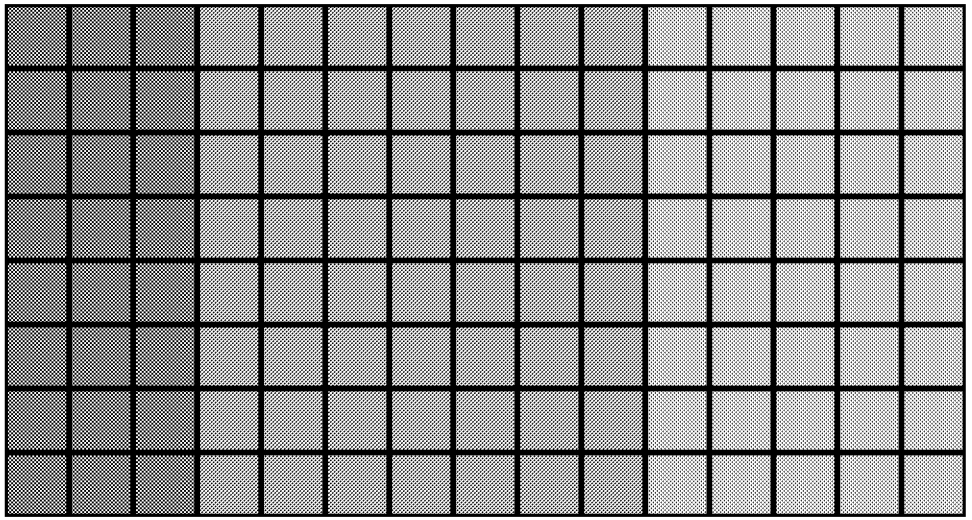
# HPF-2: Distribution with Shadowing

!HPF$ DISTRIBUTE A(*,BLOCK(SHADOW=1))



# HPF-2: Distribution with Flexible Blocksize

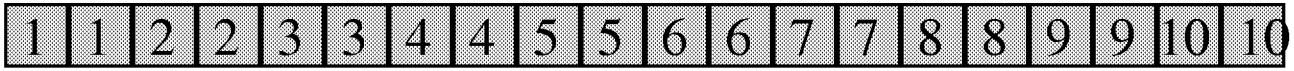
!HPF$ DISTRIBUTE A(*,BLOCK( (/3,7,5/) )



## HPF-2: Asynchronous I/O

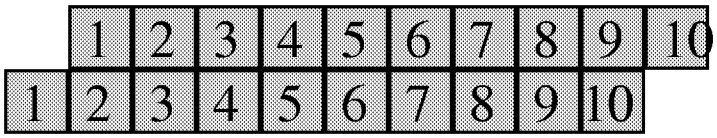
Standard

```
DO I = 1,10,1
        READ (FILE) A(I,1:1000)
        CALL COMPUTE ( A(I,1:1000))
END DO
```




Async

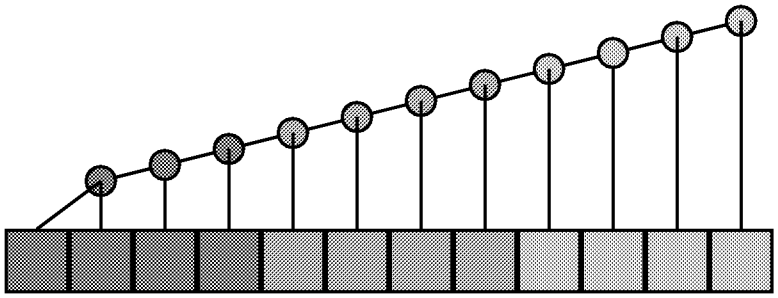
```
READ (FILE, ID=f1) A(1,1:1000)
DO I = 2,10,1
        WAIT (ID=f1)
        READ (FILE,ID=f1) A(I,1:1000)
        CALL COMPUTE ( A(I-1,1:1000))
END DO
CALL COMPUTE (A(10,1:1000)
```



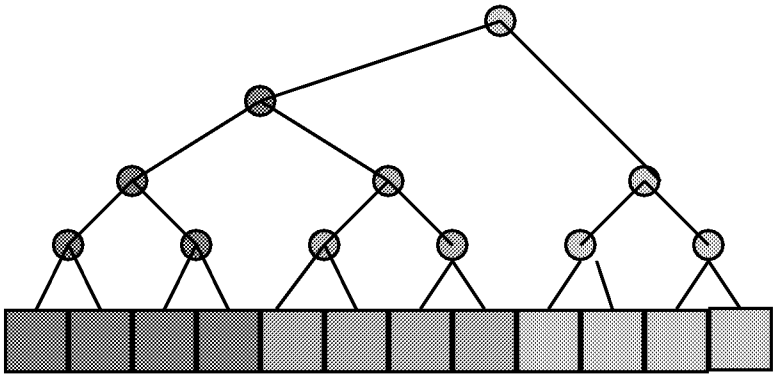
## HPF-2: Parallelism: Loop-Reductions

```
!HPF$ INDEPENDENT, NEW(c), REDUCTION(x)
      DO i = 1,n,1
         c = f(i)
         x = x + g(c)
END DO
```

Reduction-Treatment:



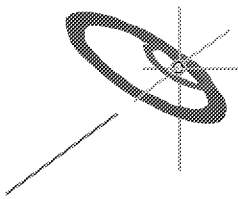
sequential (reproducable!)



parallel (tree)

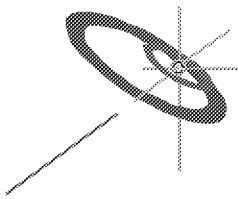
## HPF-2: Parallelism: Macrotasking

```
!HPF$       TASKING
!HPF$             ON HOME (A(1:5))
                        <statement 1>
                            :
                        <statement k>
!HPF$             ON HOME (A(5:9))
                        <statement 1>
                            :
                        <statement l>
!HPF$             ON HOME (A(10:20))
                        CALL sub()
!HPF$       END
```



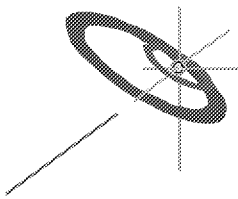
## HPF-2: Approved Extensions

- DYNAMIC, REALIGN, REDISTRIBUTE
  - Removed from baseline HPF-2 due to complexity of implementation
- HPF_SPMD (HPF_CRAFT)
  - Worksharing-model based on Cray's CRAFT programming model



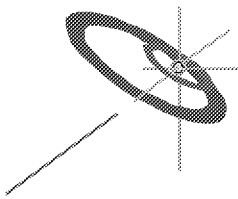
## HPF-2: Deletions and Changes

- Mapping together with sequence-association
- Mapping-change at subroutine-boundaries only allowed with explicit interface
- Other changes in the Spirit of FORTRAN-90 and FORTRAN-2000



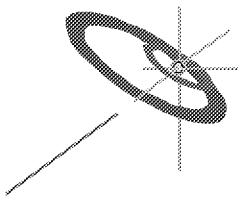
# Usability of HPF in CFD

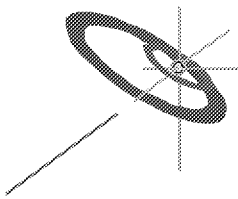
| Numerical Method | HPF-1 | HPF-2 | HPF-2 + Ext. | Comment |
|---|---|---|---|---|
| **structured** | + | + | + | High overhead in HPF-1: Missing Shadows |
| **block-structured** | - | + | + | HPF-1: Missing mapping of derived types and general block-distributions |
| **explicit, unstructured** | + | + | + | High overhead in HPF-1: Missing shadows and indirect mapping |
| **implicit, unstructured** | - | + | + | HPF-1: Missing indirect mapping |
| **FEM** | - | + | + | HPF-1: Missing indirect mapping |
| **Adaptive, structured** | + | - | + | Realign and Redistribute abandonned in HPF-2 |
| **Adaptive, unstructured** | - | - | + | Missing indirect mapping in HPF-1, Realign and Redistribute abandonned in HPF-2 |



# HPF-Products

- Applied Parallel Research (APR
- Portland Group (PGI)
- Pacific Sierra Research
- Cray
- intel
- IBM
- DEC
- Hitachi
- NEC
- Fujitsu
- Meiko
- Convex
- SGI



## HPF-Info

- WWW
  - http://www.crpc.rice.edu/HPFF/home.html
  - http://www.vcpc.univie.ac.at/HPFF/home.html
- FTP
  - titan.cs.rice.edu
  - public/HPFF/draft