Evaluation of High Performance Fortran for an Industrial CFD Code

Thomas Brandes and Falk Zimmermann
GMD-National Research Center for Computer Science
Institute for Algorithms and Scientific Computing
Schloß Birlinghoven, D-53757 St. Augustin, Germany
E-mail: `{brandes,falk.zimmermann}@gmd.de`

Christian Borel and Marc Brédif
MATRA-BAe Dynamics France, CFD Group
Service Aerodynamique Numerique PV21
37, avenue Lois Bréguet, F-78146 Vélizy-Villacoublay, France
E-mail: `{cborel,mbredif}@matra-def.fr`

VECPAR'98

Porto, June 1998

Porting of the Industrial CFD Code AEROLOG to HPF

Thomas Brandes and Falk Zimmermann
GMD-National Research Center for Computer Science
Institute for Algorithms and Scientific Computing
Schloß Birlinghoven, D-53757 St. Augustin, Germany
E-mail: `{brandes,falk.zimmermann}@gmd.de`

Christian Borel and Marc Brédif
MATRA-BAe Dynamics France, CFD Group
Service Aerodynamique Numerique PV21
37, avenue Lois Bréguet, F-78146 Vélizy-Villacoublay, France
E-mail: `{cborel,mbredif}@matra-def.fr`

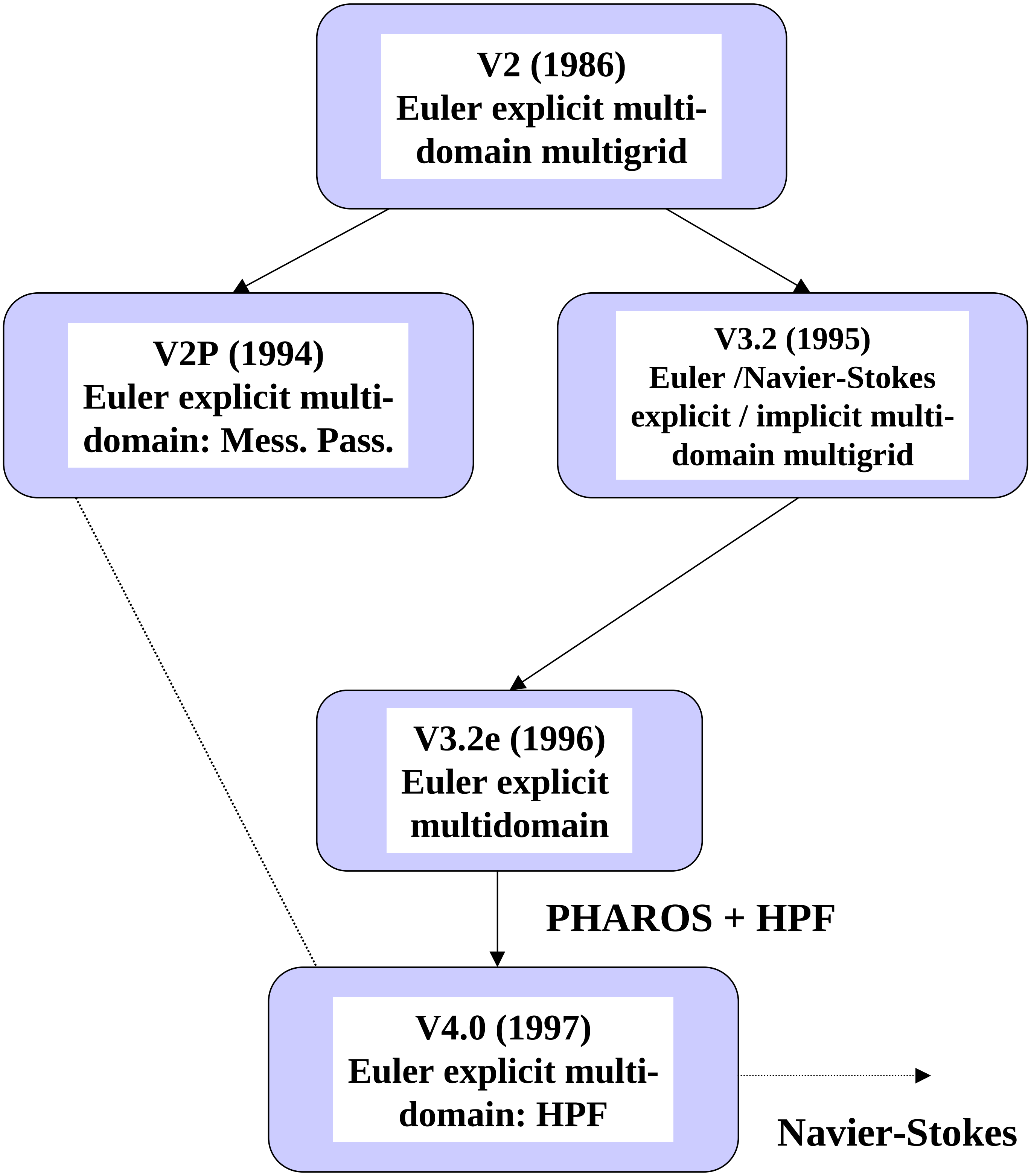
HUG'98
2nd Annual HPF User Group meeting

Porto, June 1998

AEROLOG Code

- **Proprietary software of MATRA BAe Dynamics**
- **CFD software devoted to the study of compressible fluid flows around complex geometries**
- **applied during the development programs MISTRAL, MICA, and APACHE**
- **reduced the experimental studies and consequently cutting down costs and delays**

About AEROLOG



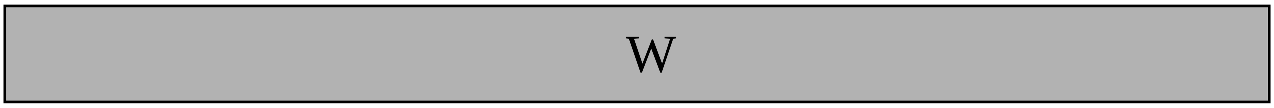
Versions of AEROLOG

```fortran
program AEROLOG
...
call CLAIND   ! Dynamic partitioning
call CLASS0   ! Reading values
call CLASS1   ! Computations
...
end program AEROLOG


subroutine CLASS1
...
call CLASTA
TIME_LOOP: do IT =NITMIN, NITMAX
   call CLSCHM (...)
   if (condition) then
      call CLWRRS (...)
      call MODLISA(...)
      call CLWRTR (...)
   end if
end do TIME_LOOP
....
end subroutine CLASS1
```

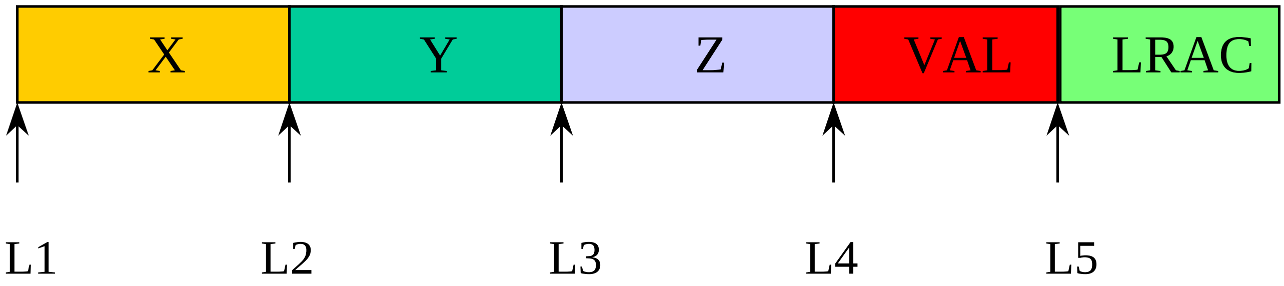
- 55 subroutines
- 11500 lines of source code

V3.2e: Code Structure (1)

```
real W(100000)          ! Static Workspace
```



Dynamic Partitioning:




```
call CLAIND(...,L1,L2,L3,L4,L5,...)
...
call CLASS1(...,W(L1),W(L2),W(L3),...)

subroutine CLASS1(...,X,Y,Z,...)
real, dimension (NTOT) :: X, Y, Z
...
```

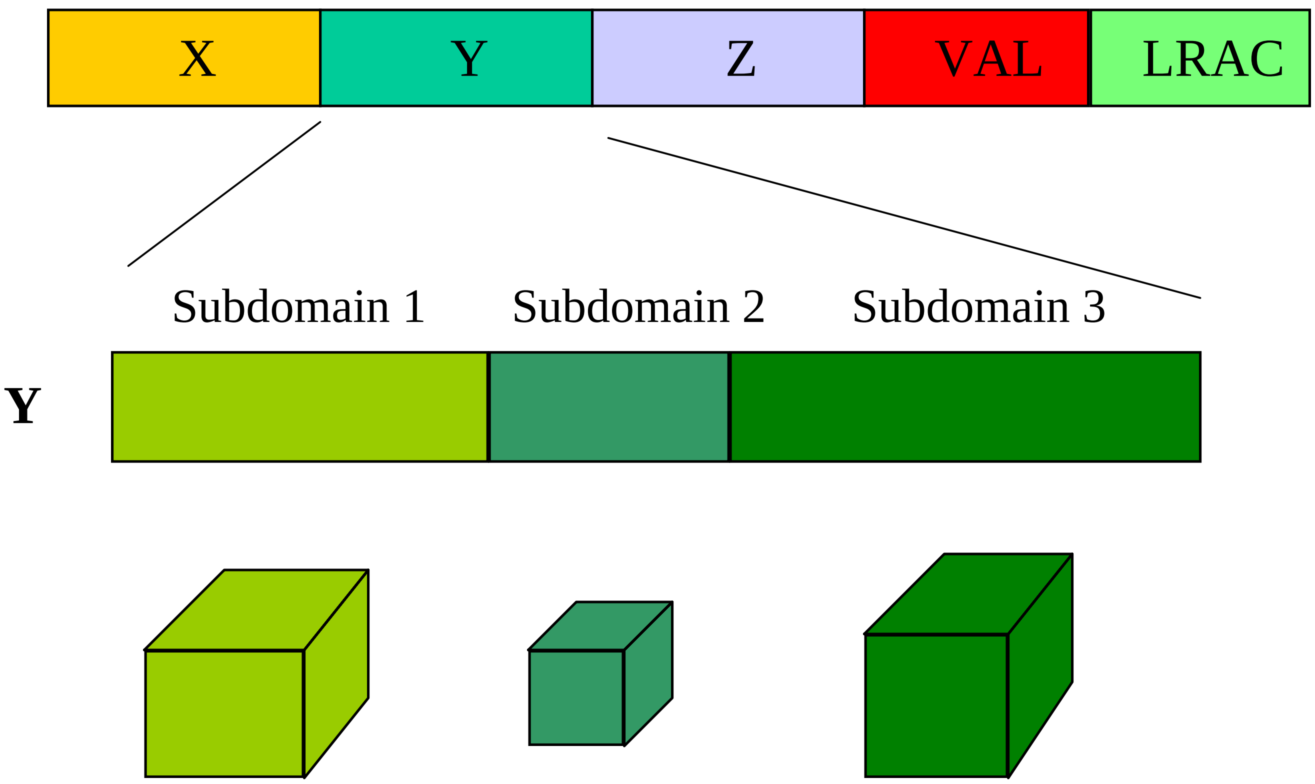
V3.2e: Data Structure (Level 1)

```
subroutine CLSCHM (...)
...
do NSD=1,NSDTOT
   call CLTIME (..)   ! Time step
   call CLSCH1 (..)   ! Non viscous Dt fluxes
   call CLAVIS (..)   ! Artificial Viscosity Coeff.
end do
call CLRACV (..)      ! Matching of artif. Viscosity coef.
call CLCHB1 (..)      ! Flux Boundary Conditions
do NSD=1,NSDTOT
   call CLSCH2 (..)   ! Viscous and Dt^2 fluxes
   call CLVISC (..)   ! Artificial viscosity terms
   call CLFLIN (..)   ! Fluxes increments
end do
call CLRACF (..)      ! Matching of field increments
call CLCHB0 (..)      ! Boundary Conditions of field values
do NSD=1,NSDTOT
   call CLCVRF (..)   ! Update of field values
end do
call CLBDRF (..)      ! Update boundary values
...
end subroutine CLSCHM
```

- **local routines (around 90% of computation time)**
- **boundary routines**

V3.2e: Code Structure (2)

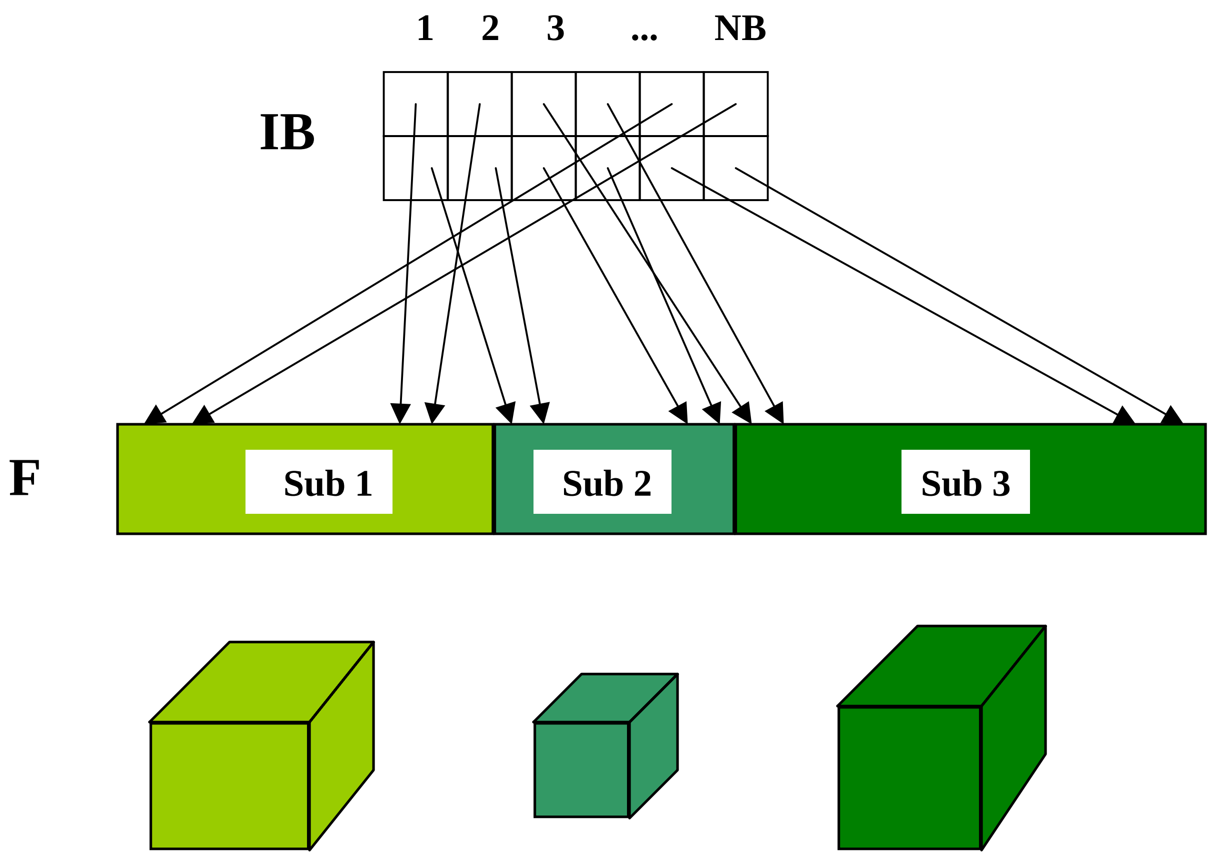
Linearization of Data




```
POINTER=0
do NSD=1,NSDTOT
  call LOCAL_SUB (IMAX(NSD),JMAX(NSD),
                  KMAX(NSD),Y(POINTER))
  POINTER=POINTER+IMAX(NSD)*JMAX(NSD)*KMAX(NSD)
end do

subroutine LOCAL_SUB (IM, JM, KM, Y, ...)
integer IM, JM, KM
real Y(IM, JM, KM)
...
```

V3.2e: Data Structure (Level 2)

```
subroutine BOUNDARY (F, NTOT, IB, NB)
integer, intent(in) :: NTOT, NB
real, dimension (NTOT), intent(inout) :: F
integer, dimension (2,NB), intent(in) :: IB

do I=1,NB
   X1 = F(IB(1,I))
   X2 = F(IB(2,I))
   X = (X1+X2) * 0.5
   F(IB(1,I)) = X
   F(IB(2,I)) = X
end do

end subroutine BOUNDARY
```



Computation of Boundary Conditions

Validation Test Cases

Test Case Name	Size x Subdomains	Total Mesh Points	Processing nodes	Memory / node (MB)
Small_1	(65 x 33 x 9) x 1	19305	1	4
Small_2	(65 x 33 x 9) x 2	38610	2	4
Small_4	(65 x 33 x 9) x 4	77220	4	4

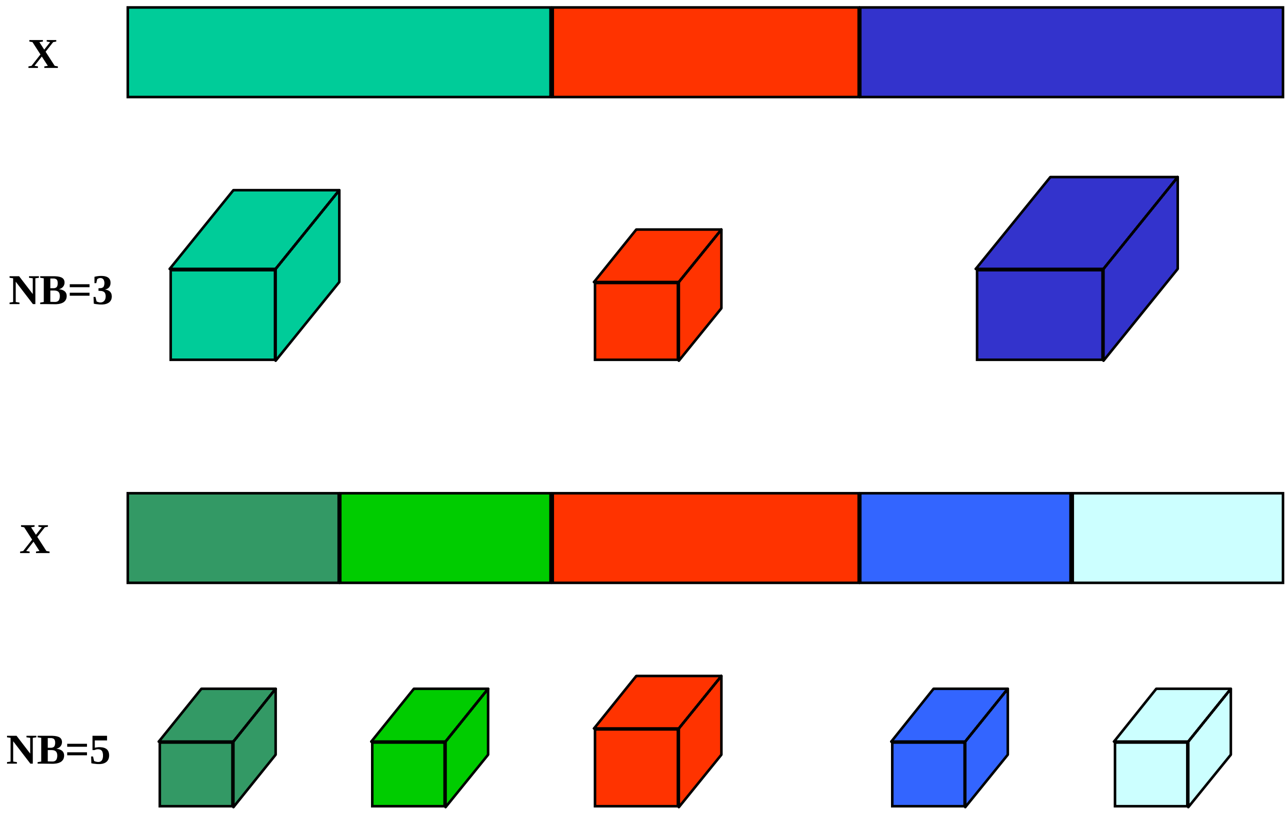
Industrial Test Cases

Test Case Name	Size x Subdomains	Total Mesh Points	Processing nodes	Memory / node (MB)
Small_8	(65 x 33 x 9) x 8	154400	8-16	4-2
Medium_8	(65 x 65 x 9) x 8	304200	16-32	4-2
Large_8	(129 x 65 x 9) x 8	603720	32-64	4-2

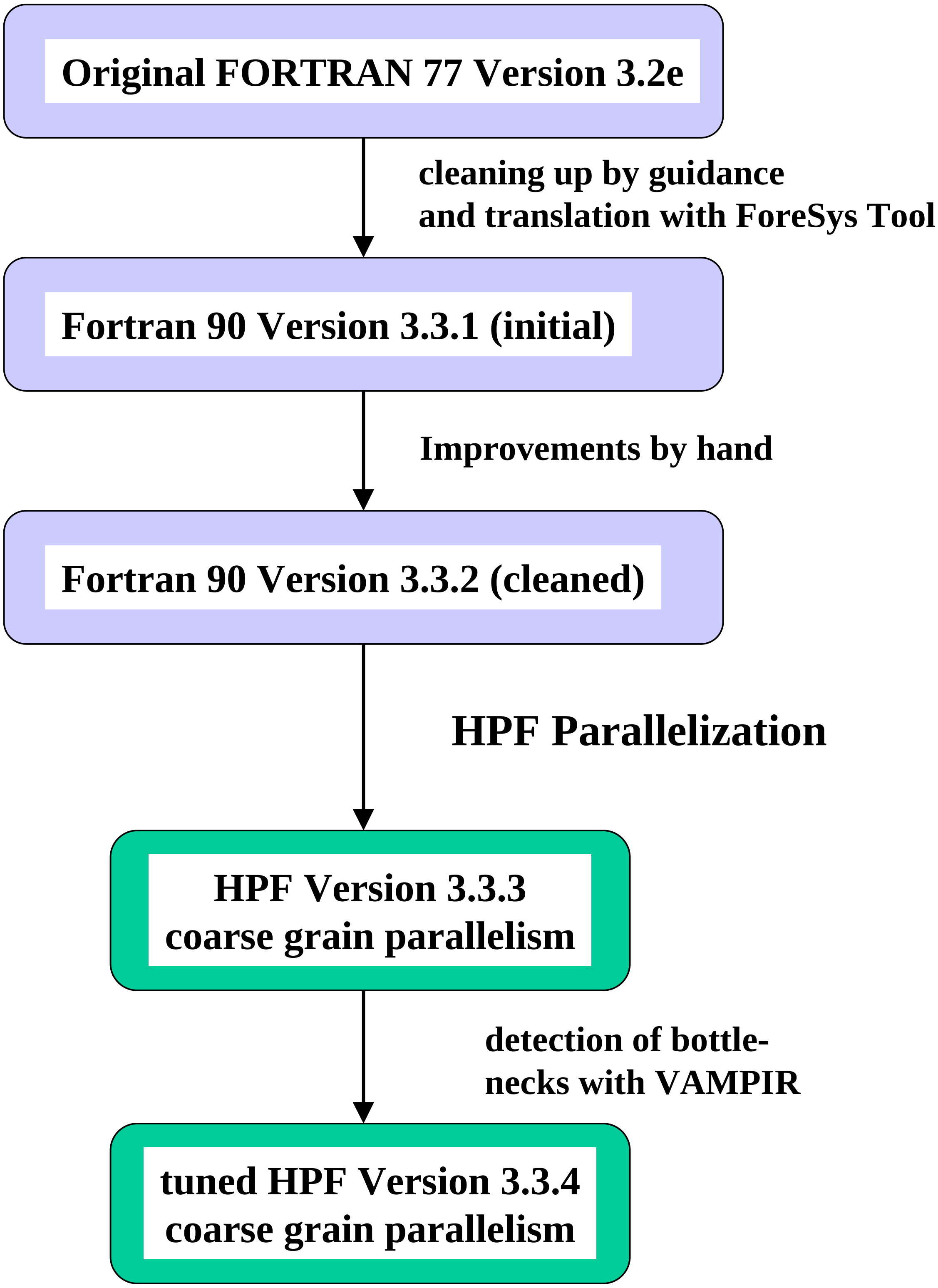
- **All cases based on the same blunt body geometry, supersonic flow**

Aerodynamic Test Cases

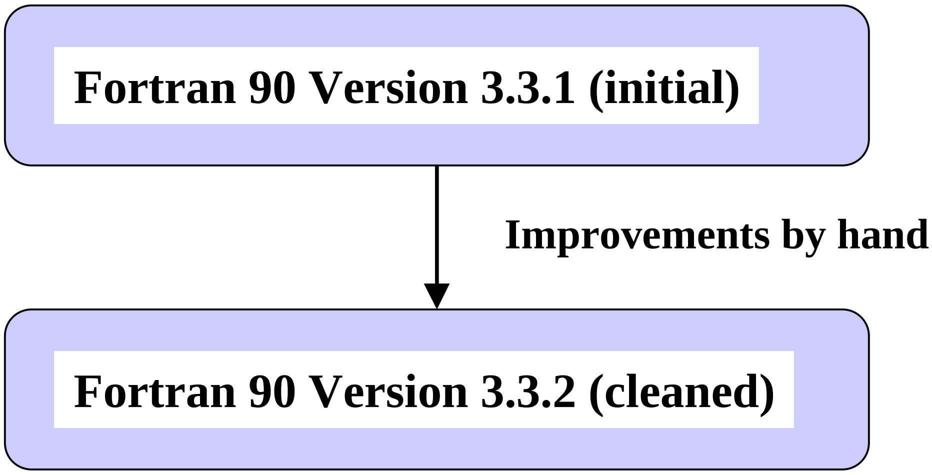
Tool for Splitting Subdomains available



Artifical Subdomains



Versions of AEROLOG



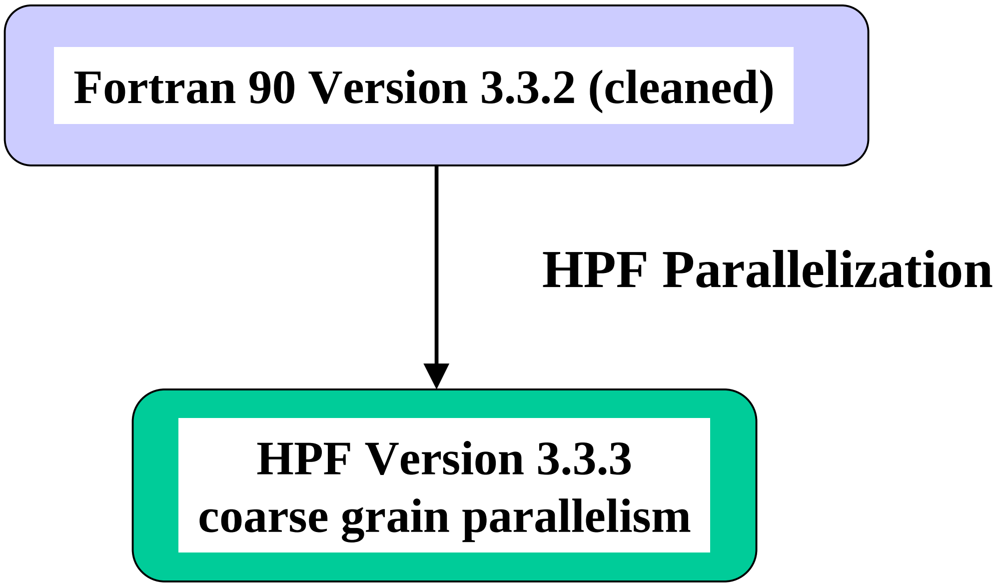
- **Super array replaced with allocatable / automatic arrays**
- **Assumed size arrays removed**
- **Loops into array operations**
- **maximal sized arrays adapted to actual problem size**
- **correction of dummy list arguments**

Porting to Fortran 90

Super array replaced with allocatable and automatic arrays !


```fortran
...
allocate (X(NTOT),Y(NTOT),Z(NTOT))
call CLASS1(..., X, Y, Z, M,...)
...

subroutine CLASS1(..., X, Y, Z, M, ..)
real, dimension (NTOT) :: X, Y, Z
real, dimension (M) :: VAL      ! automatic
integer, dimension (M) :: LRAC ! arrays
...
end subroutine
```

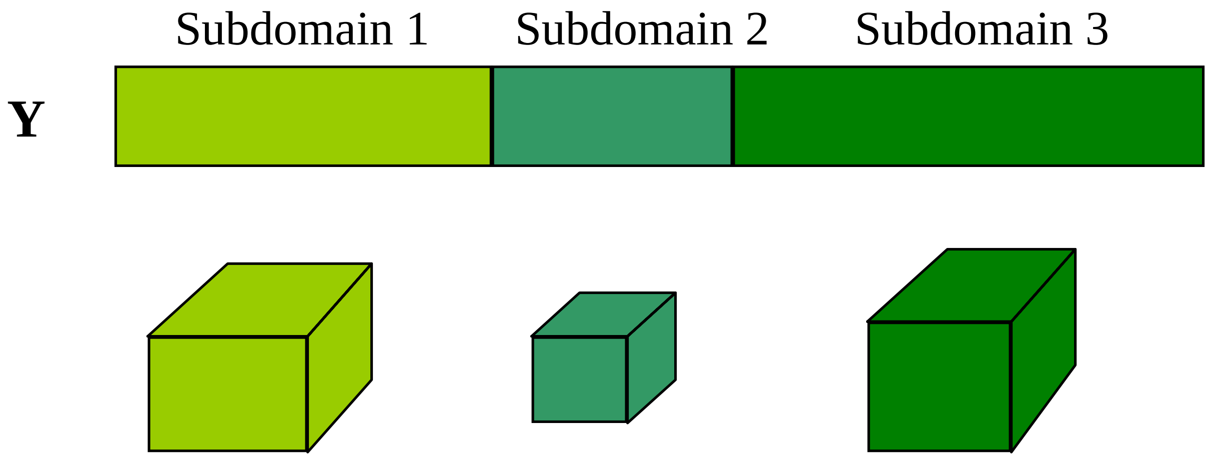
Introduction of Dynamic Arrays



- **Loops over subdomains provide coarse grain parallelism without communication, communication only in boundary calculations**
- **Fine grain parallelism by distribution of every subdomain (ideas welcome!)**

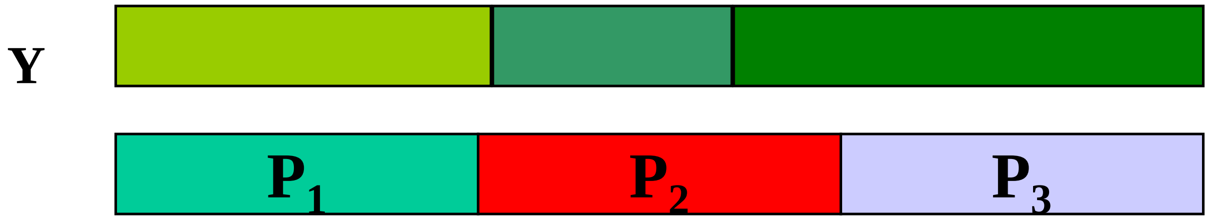
Main problem: How to specify the data distribution ???

HPF Parallelization



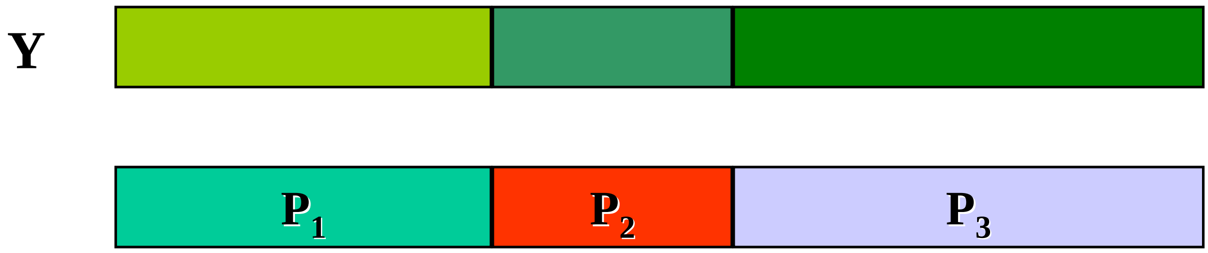
Not possible:

```
      real, dimension (NTOT)   :: Y
!hpf$ distribute (BLOCK) :: Y
```



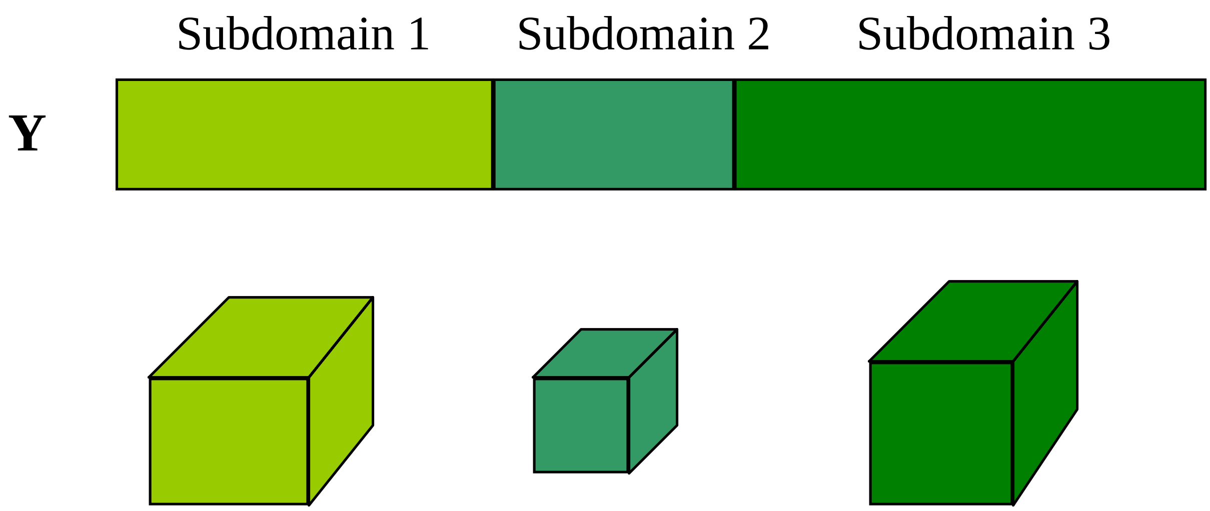
Unsupported (HPF 2.0) in commercial HPF compilers:

```
      real, dimension (NTOT)        :: Y
!hpf$ distribute (GEN_BLOCK(SIZES)) :: Y
```

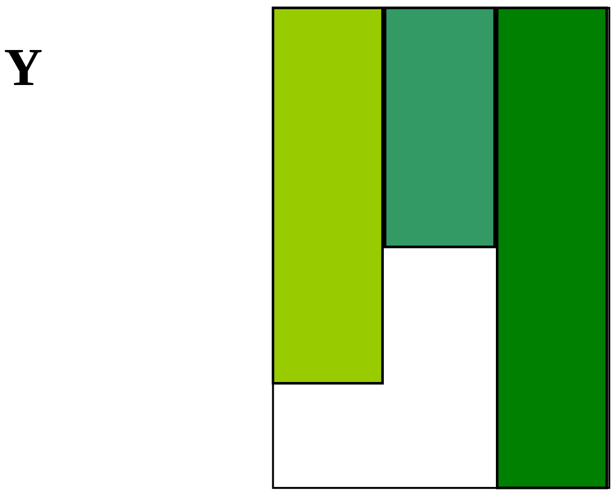


Distribution of Subdomains (1)

Our Approach:



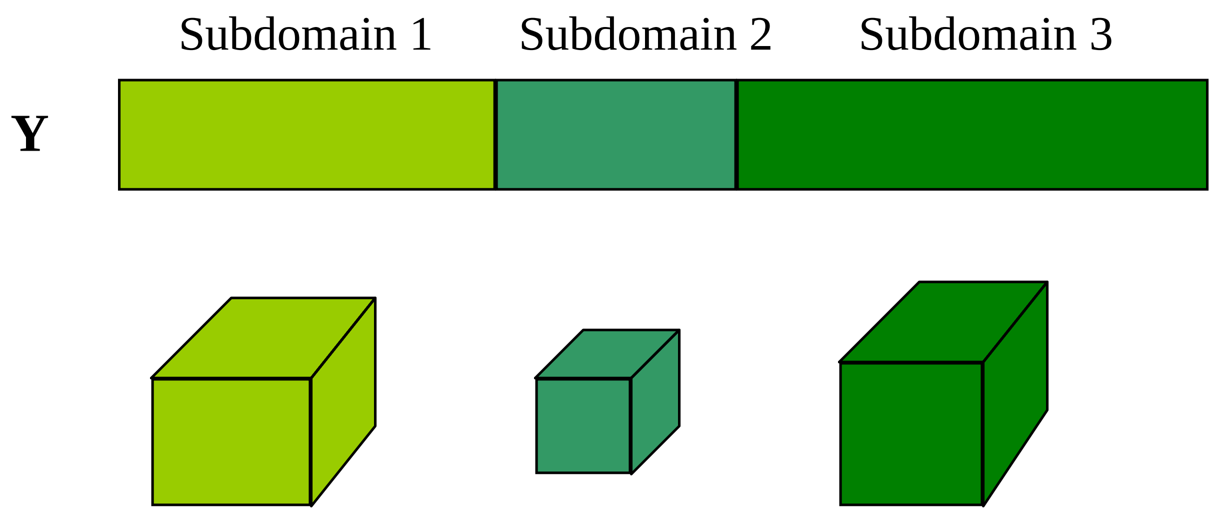

```
      real, dimension (NMAX,NSDTOT)   :: Y
!hpf$ distribute (*,BLOCK) :: Y
```



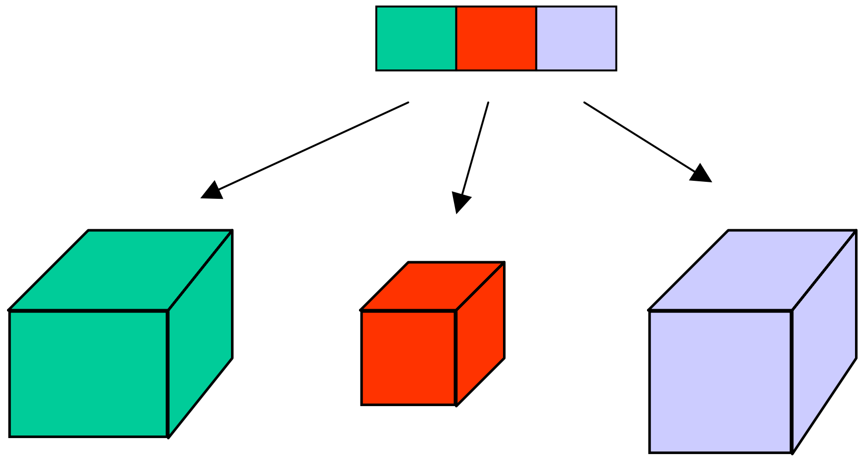
Attention: still relies on sequence association for a subdomain

Distribution of Subdomains (2)

The ‘good’ Solution:




```
type SUBDOMAIN
   integer :: IM, JM, KM
   real, allocatable (:,:,:) :: X, Y, Z
end type

type (SUBDOMAIN), allocatable (:) :: SUBDOMAINS
!hpf$ distribute SUBDOMAINS (BLOCK)
```



Distribution of Subdomains (3)

HPF Parallelization of Local Computations:

Not possible:

```
      real, dimension (NMAX,NSDTOT)   :: X, Y, Z
!hpf$ distribute (*,BLOCK) :: X, Y, Z

!hpf$ independent
      do NSD = 1, NSDTOT
         call CLTIME (IM(NSD), JM(NSD), K(NSD),
     &                   X(:,NSD), Y(:,NSD),
     &                   Z(:,NSD), ...)
      end do

      pure subroutine CLTIME (IM, JM, KM, X, Y, Z, ...)
      integer, intent(in) :: IM, JM, KM
      real, dimension (IM, JM, KM) :: X, Y, Z
      ...
      end subroutine CLTIME
```

Problem: Sequence Association

HPF Parallelization of Local Computations:

PGI Solution:

```
      real, dimension (NMAX,NSDTOT)   :: X, Y, Z
!hpf$ distribute (*,BLOCK) :: X, Y, Z

!hpf$ independent
      do NSD = 1, NSDTOT
         call CLTIME (IM(NSD), JM(NSD), K(NSD),
     &                X(1,NSD), Y(1,NSD),
     &                Z(1,NSD), ...)
      end do

      pure subroutine CLTIME (IM, JM, KM, X, Y, Z, ...)
      integer, intent(in) :: IM, JM, KM
      real, dimension (IM, JM, KM) :: X, Y, Z
!hpf$ sequence :: X, Y, Z
      ...
      end subroutine CLTIME
```

HPF Parallelization of Local Computations:

NAS HPFPlus Solution:

```
      real, dimension (NMAX,NSDTOT)   :: X, Y, Z
!hpf$ distribute (*,BLOCK) :: X, Y, Z

      do NSD = 1, NSDTOT
         call CLTIME (IM(NSD), JM(NSD), K(NSD),
     &                X(1,NSD), Y(1,NSD),
     &                Z(1,NSD), ...)
      end do

! For the interface:

      extrinsic (HPF_SERIAL)
     & subroutine CLTIME (IM, JM, KM, X, Y, Z, ...)
      integer, intent(in) :: IM, JM, KM
      real :: X, Y, Z
      ...
      end subroutine CLTIME

! But compile with F77 compiler:

      subroutine CLTIME (IM, JM, KM, X, Y, Z, ...)
      integer, intent(in) :: IM, JM, KM
      real, dimension (IM,JM,KM) :: X, Y, Z
      ...
      end subroutine CLTIME
```

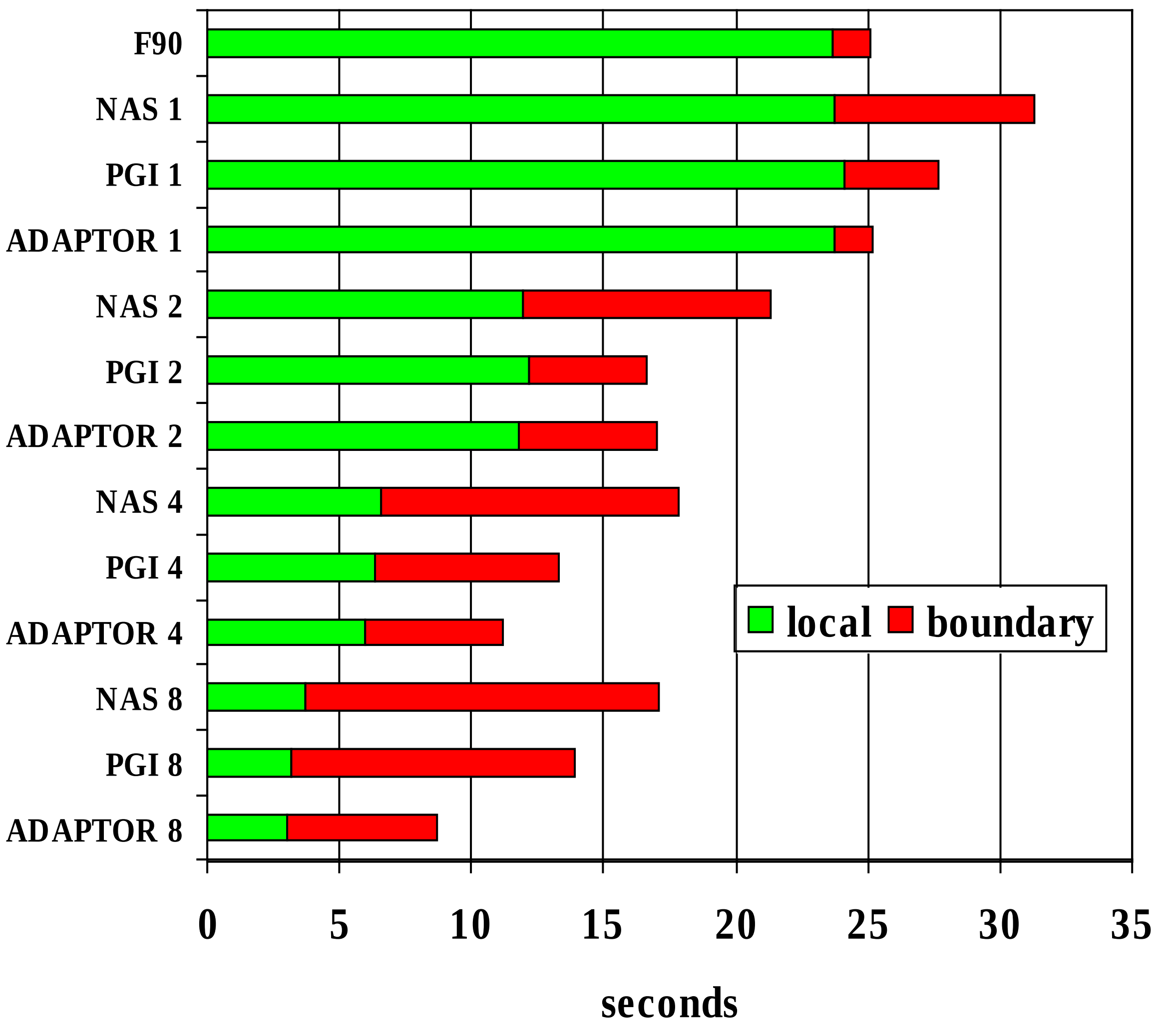
```
subroutine BOUNDARY (F, NMAX, NSD, ISD_B, IJK_B, NB)
integer, intent(in) :: NMAX, NSD, ISD_B, IJK_B
real, dimension (MAX,NSD), intent(inout) :: F
integer, dimension (2,NB), intent(in) :: ISD_B, IJK_B

do I=1,NB
   X1 = F(ISD_B(1,I),IJK_B(1,I))
   X2 = F(ISD_B(2,I),IJK_B(2,I))
   X = (X1+X2) * 0.5
   F(ISD_B(1,I),IJK_B(1,I)) = X
   F(ISD_B(2,I),IJK_B(2,I)) = X
end do

end subroutine BOUNDARY
```

- **boundary conditions are replicated on all processors**
- **implicit redistribution (replication) at subroutine boundaries utilized**

Boundary Conditions in HPF Version

Results on IBM SP2 (thin nodes), Small_8, 5 Iterations

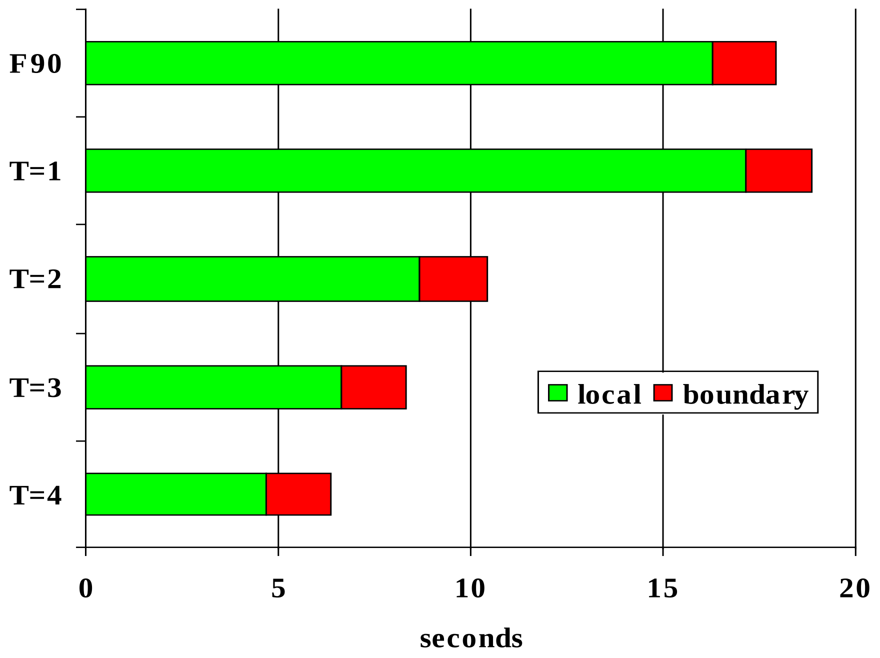
- **ADAPTOR HPF System 5.1 (SCAI, GMD)**
- **PGI HPF, AIX**
- **NAS HPFPlus, Release 2.01**

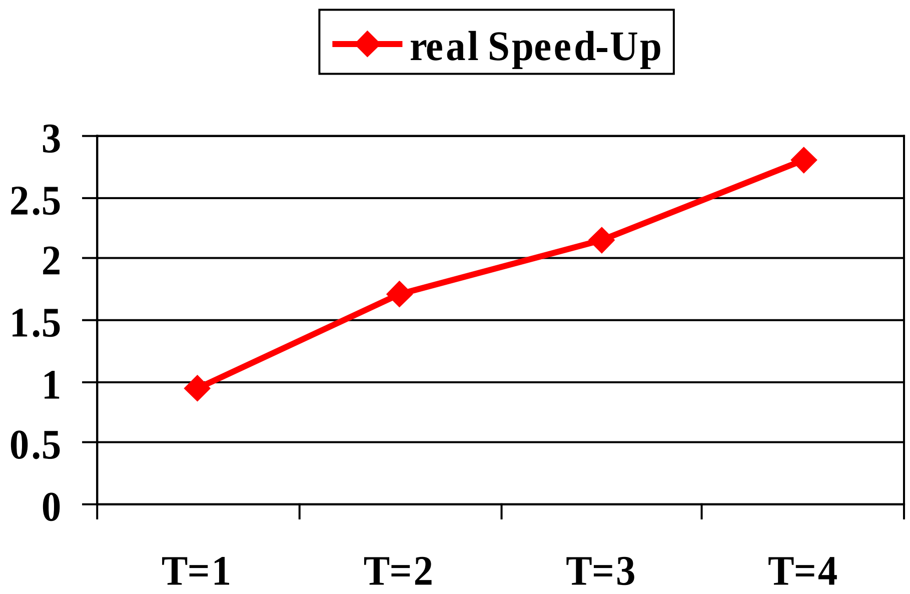


Results (Initial Version)

Results on SUN Multiprocessor, Small_8, 5 Iterations

- ADAPTOR HPF System 6.0a (SCAI, GMD)



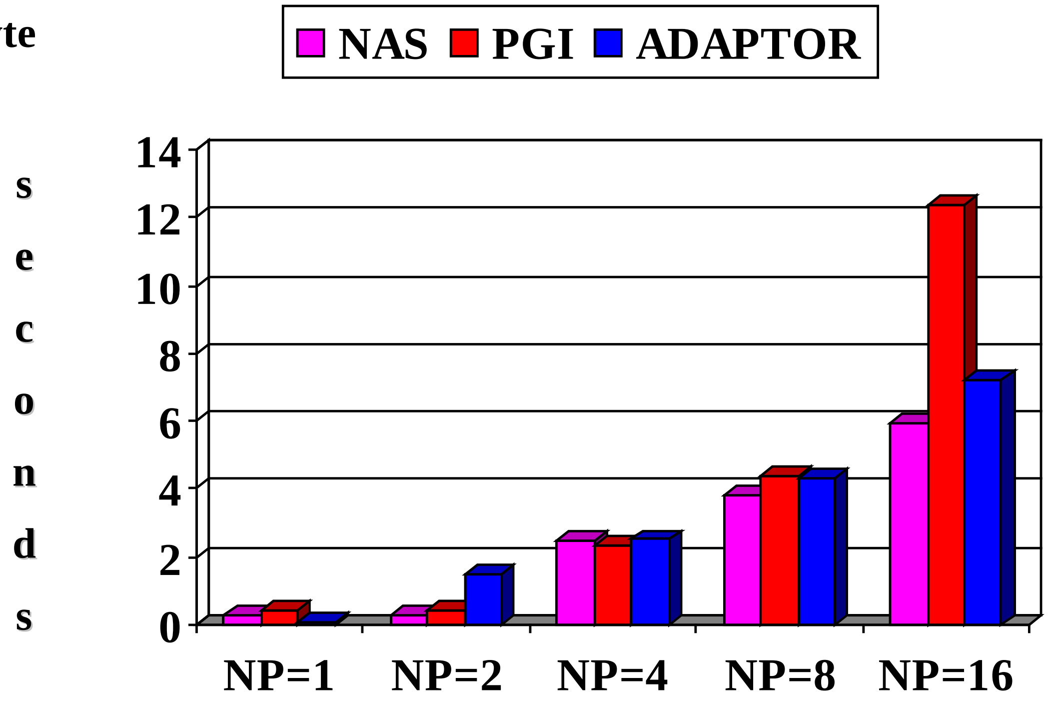


Results (Initial Version)

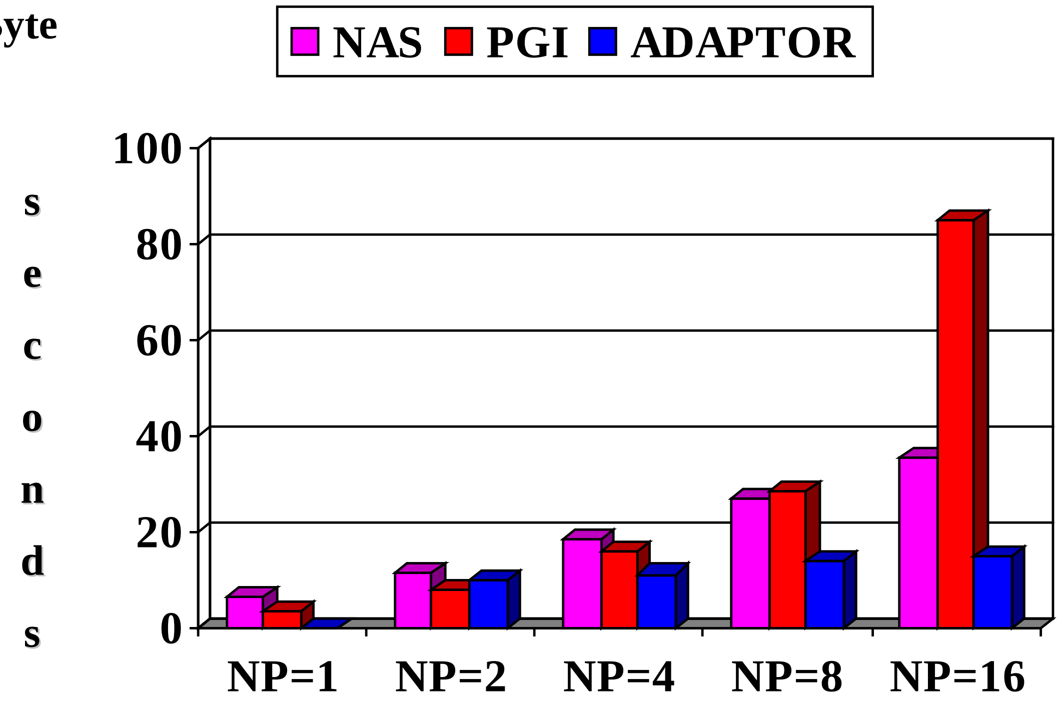
Replication of Data

```
        real, dimension (M,NP) :: A, RA
!hpf$ distribute A(*,block)
        RA = A
```

16 kByte

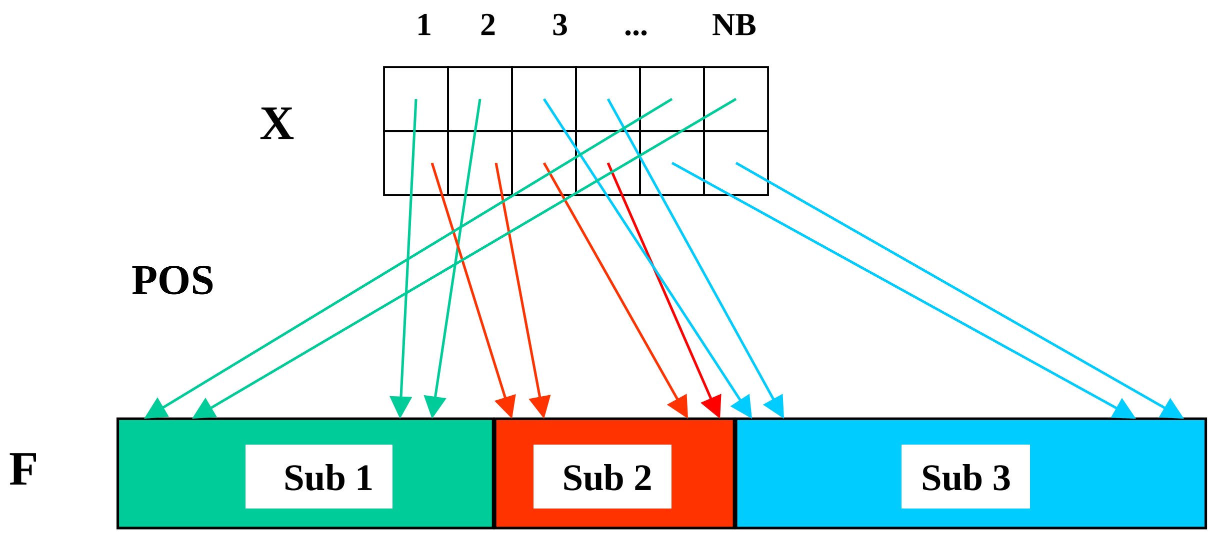


128 kByte



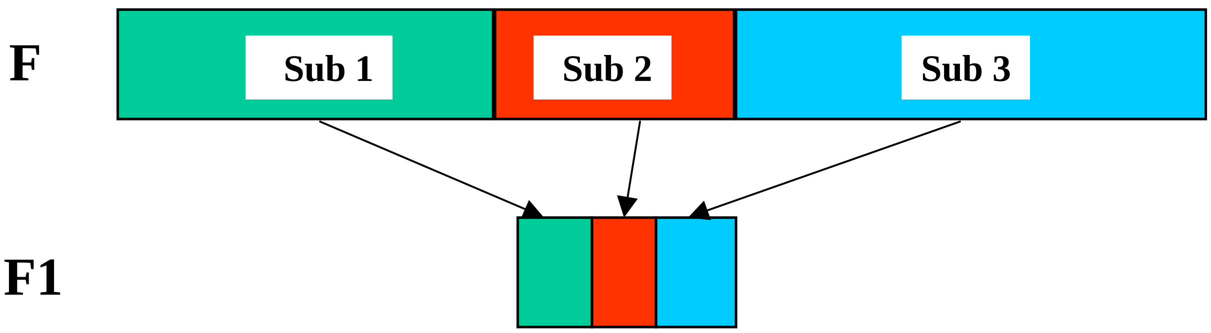
Replication of Data: Execution Times

```
      subroutine BOUNDARY (F, NMAX, NSD, ...)
      ...
      real, dimension X(NB)
      integer, dimension (2,BMAX,NSDTOT) :: POS
      X = 0.0
!hpf$ independent, reduction(X)
      do NSD=1,NSDTOT
         do K=1, NSD_B (NSD)
            X_POS = POS(1,K,NSD)
            F_POS = POS(2,K,NSD)
            X(X_POS) = X(X_POS) + F(NSD,F_POS)
         end do
      end do
      X = X * 0.5
!hpf$ independent
      do NSD=1,NSDOT
         ...
         F(NSD,F_POS) = X(X_POS)
      end do
      end subroutine BOUNDARY
```



Tuning of Boundary Computations: Reduction

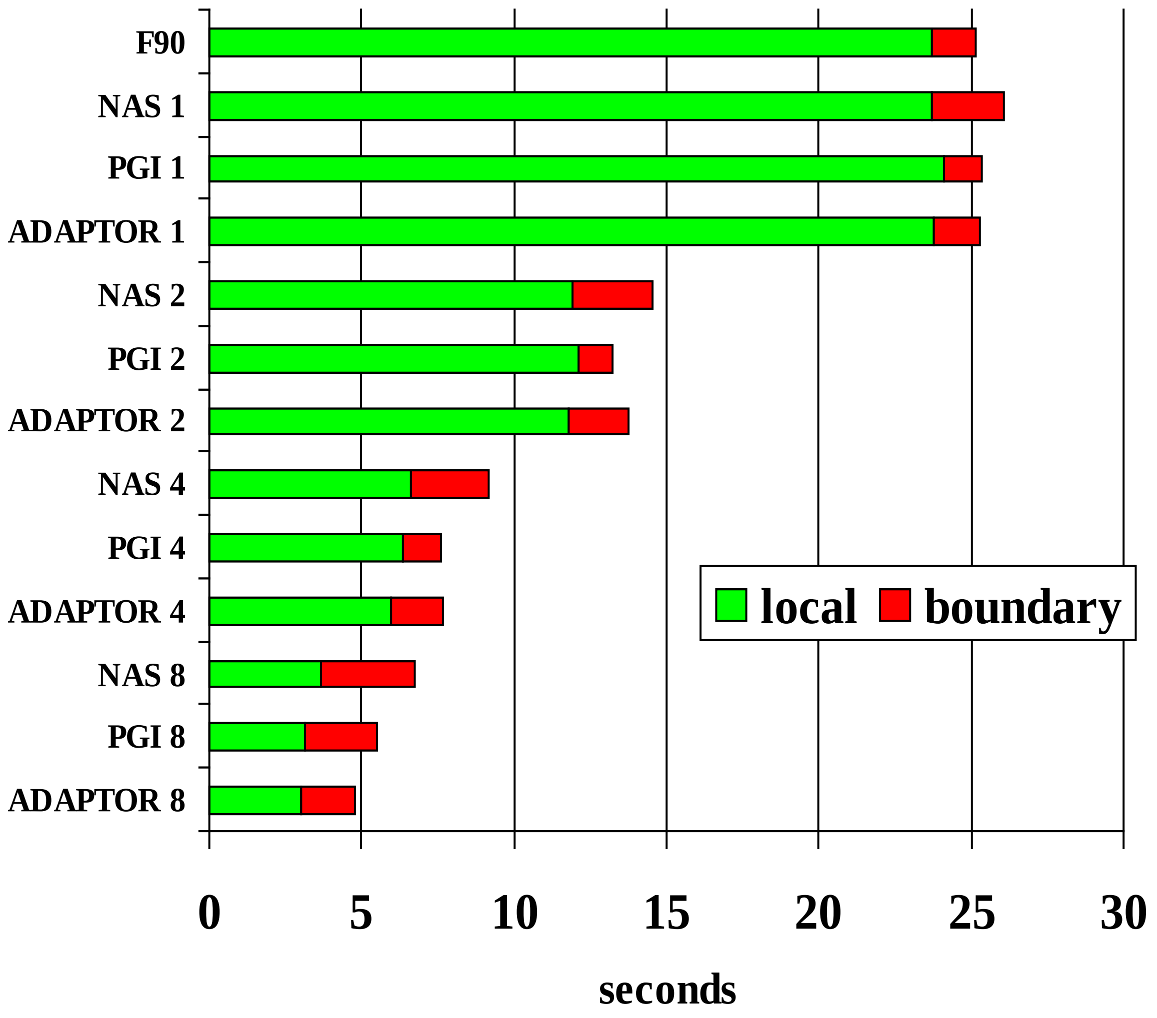
```
subroutine BOUNDARY (F, NMAX, NSDTOT, ISD_B, IJK_B, NB)
integer, intent(in) :: NMAX, NSDTOT, ISD_B, IJK_B
real, dimension (NMAX,NSDTOT), intent(inout) :: F
real, dimension (NMIN,NSDTOT) :: F1, RF1
!hpf$ distribute (*,BLOCK) :: F
integer, dimension (2,NB), intent(in) :: ISD_B, IJK_B
...
!hpf$ independent
do NSD=1,NSDTOT
   pack into F1(:,NSD)
end do
RF1 = F1
...          ! boudary computations with RF1
!hpf$ independent
do NSD=1,NSDTOT
   unpack from F1(:,NSD) into F
end do
end subroutine BOUNDARY
```



- very technical
- many new data structures
- very similar to message passing

Tuning of Boundary Conditions: Packing

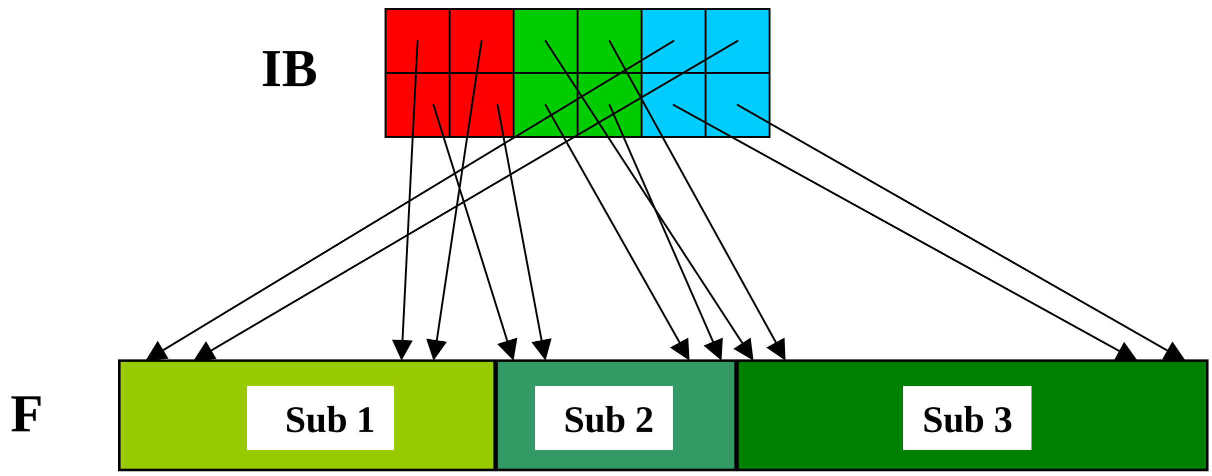
Results on IBM SP2 (thin nodes), Small_8, 5 Iterations



- **ADAPTOR HPF System 5.1 (SCAI, GMD)**
- **PGI HPF, AIX**
- **NAS HPFPlus, Release 2.01**

Results (Tuned Version)

```
subroutine BOUNDARY (F, NTOT, IB, NB)
integer, intent(in) :: NTOT, NB
real, dimension (NTOT), intent(inout) :: F
integer, dimension (2,NB), intent(in) :: IB

!hpf$ independent, new (X,X1,X2)
do I=1,NB
   X1 = F(IB(1,I))
   X2 = F(IB(2,I))
   X = (X1+X2) * 0.5
   F(IB(1,I)) = X
   F(IB(2,I)) = X
end do

end subroutine BOUNDARY
```

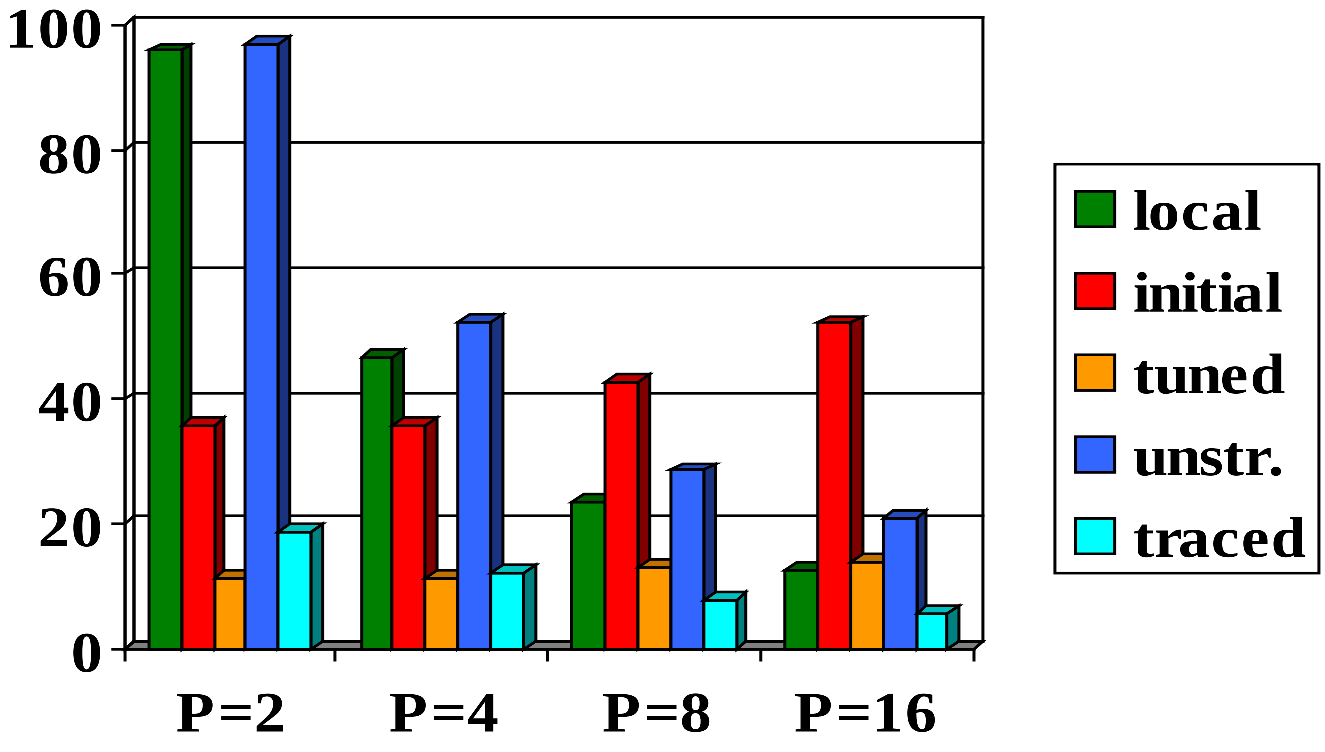


- Utilization of Inspector / Executor Scheme
- reusing of communication schedules (TRACE)

Parallelization of Boundary Conditions

Boundary Computations (ADAPTOR only)

- local routines (scale well)
- initial (replication of mesh data)
- tuned (replication of boundary mesh data)
- unstr. (unstructured communication, Insp./Exec.)
- traced (reuses communication schedule, Exec. only)

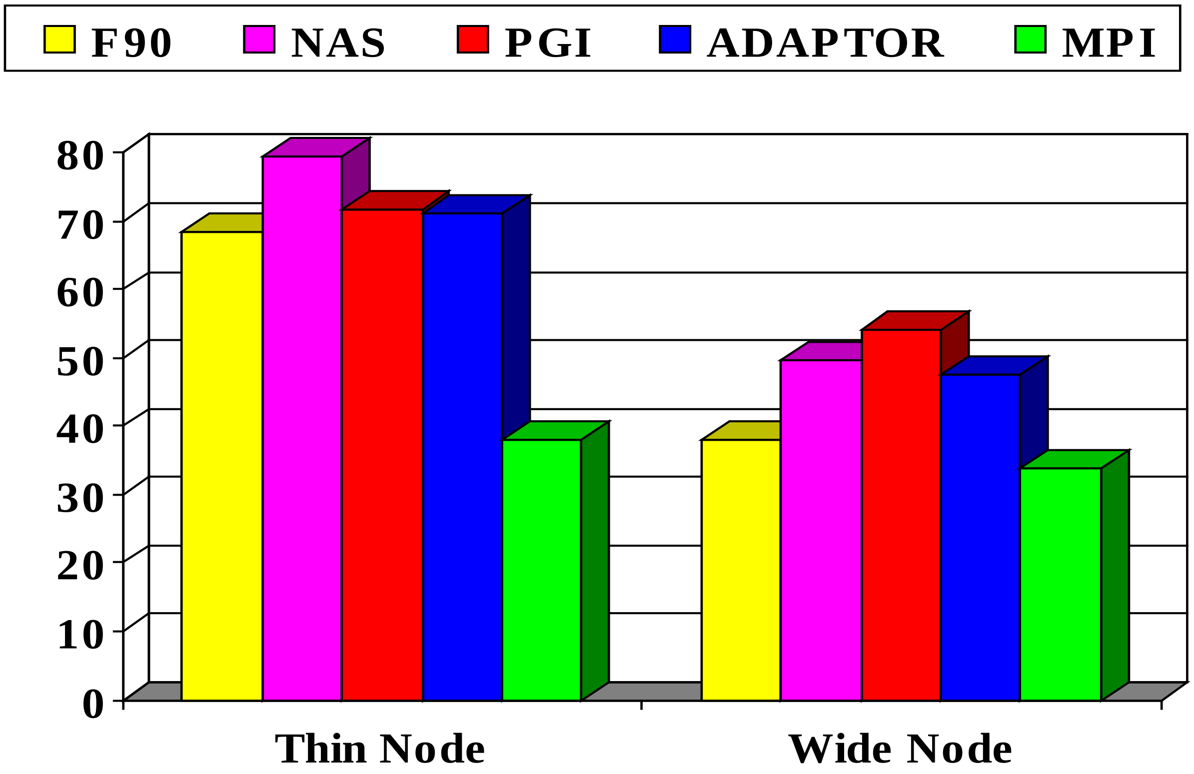


Results on IBM SP2 (thin nodes), Medium_16, 100 Iterations
All versions compiled with ADAPTOR tool.

Comparison of Boundary Computations

Comparison with MPI

- **Many code restructuring required**



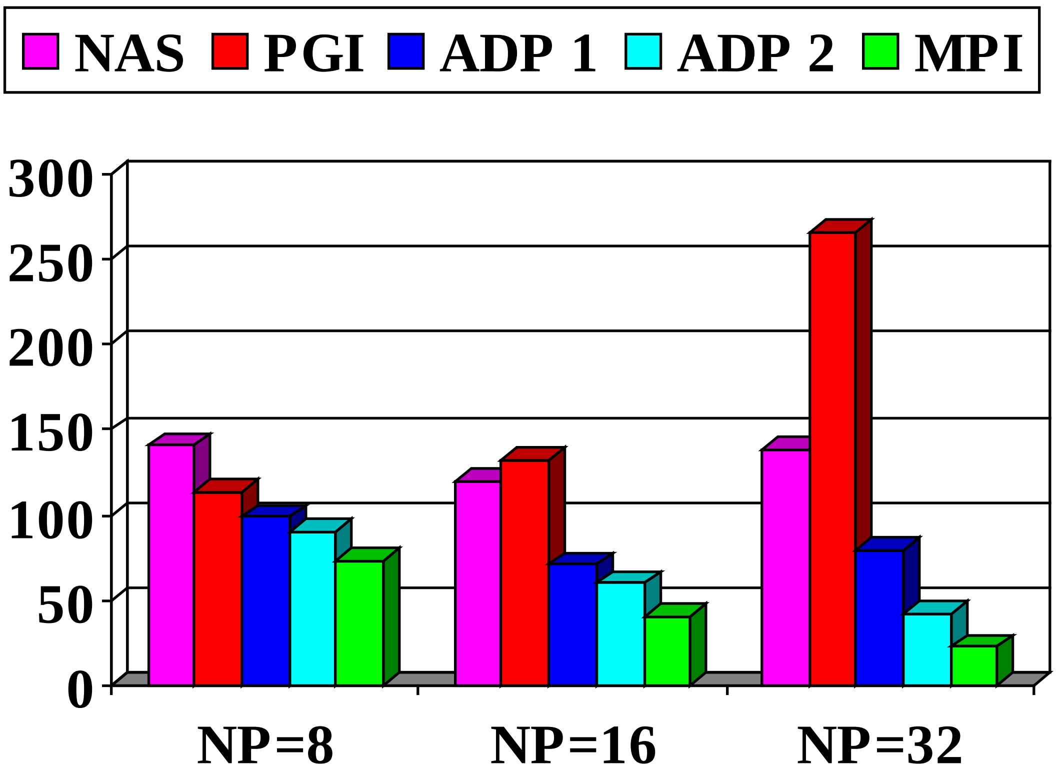
Results on IBM SP2, Small_1, 100 Iterations

But: **only two wide nodes available, so absolute comparison is not fair**

Comparison with MPI Version

Industrial Test Case 1: Small

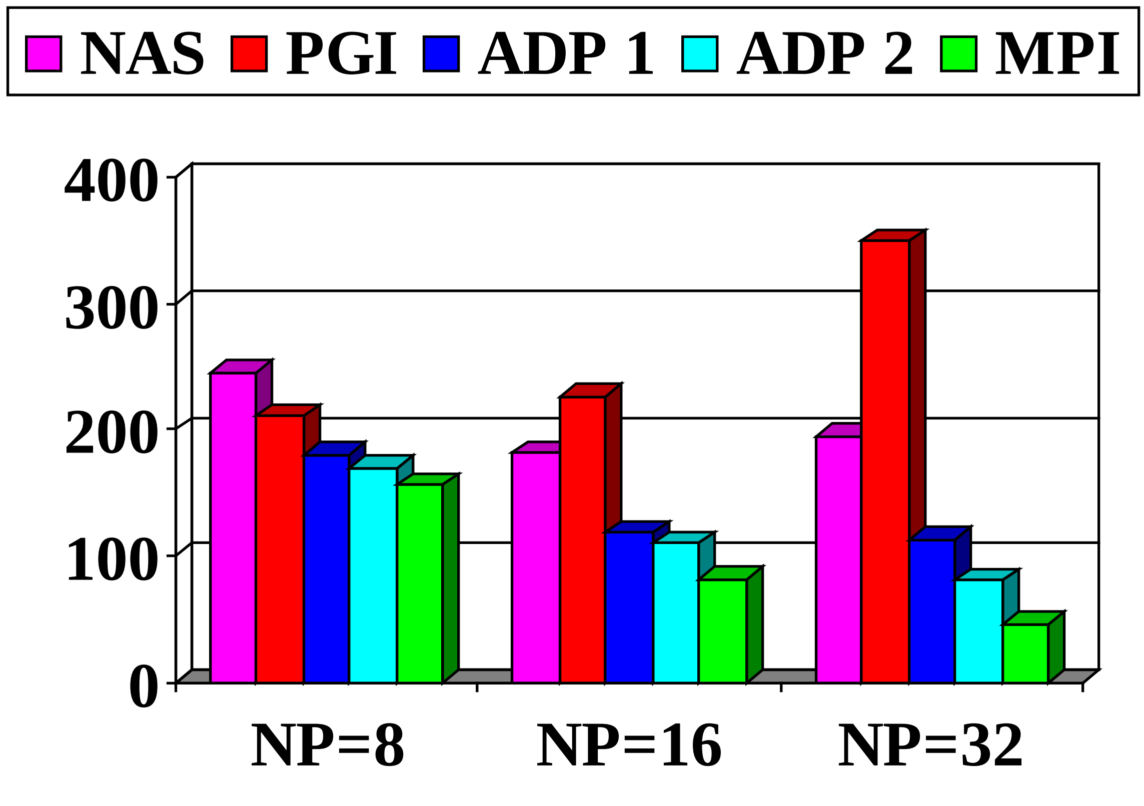
NAS:	**NAS HPFPlus 2.01**
PGI:	**PG HPF**
ADP 1:	**ADAPTOR 5.1**
ADP 2:	**ADAPTOR 6.0a (traced)**
MPI:	**Aerolog V2P**



Execution Times: SMALL

Industrial Test Case 2: Medium

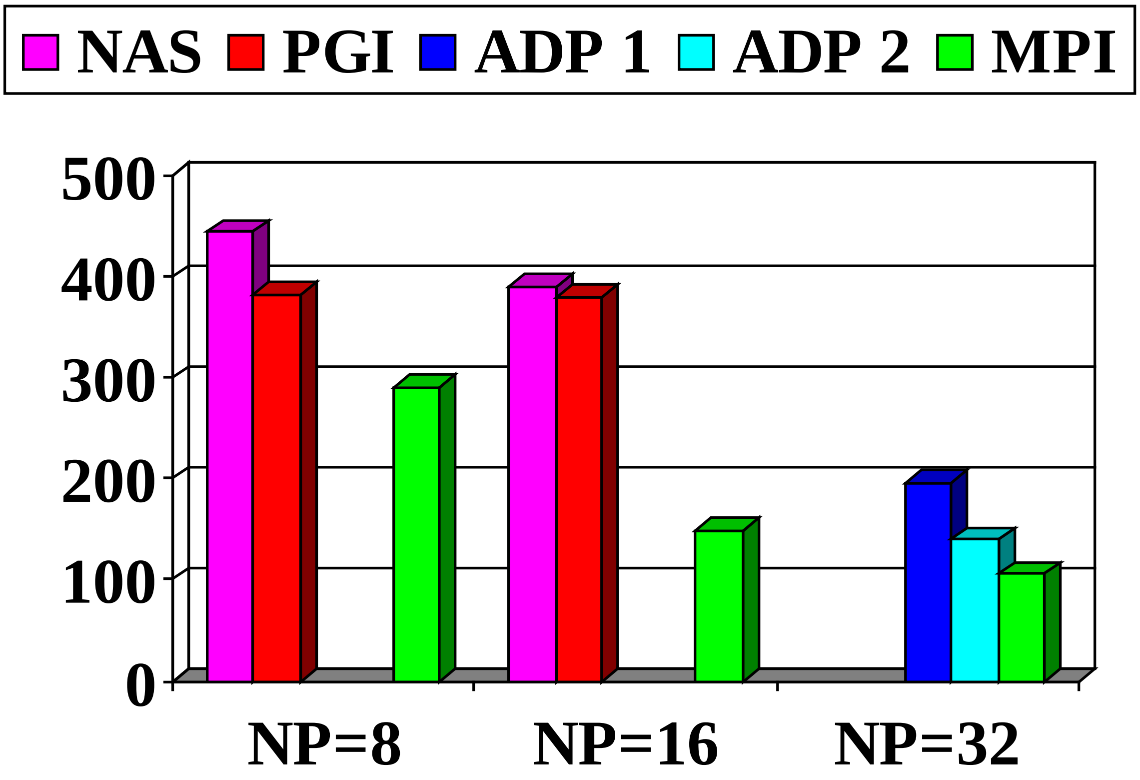
NAS:	**NAS HPFPlus 2.01**
PGI:	**PG HPF**
ADP 1:	**ADAPTOR 5.1**
ADP 2:	**ADAPTOR 6.0a (traced)**
MPI:	**Aerolog V2P**



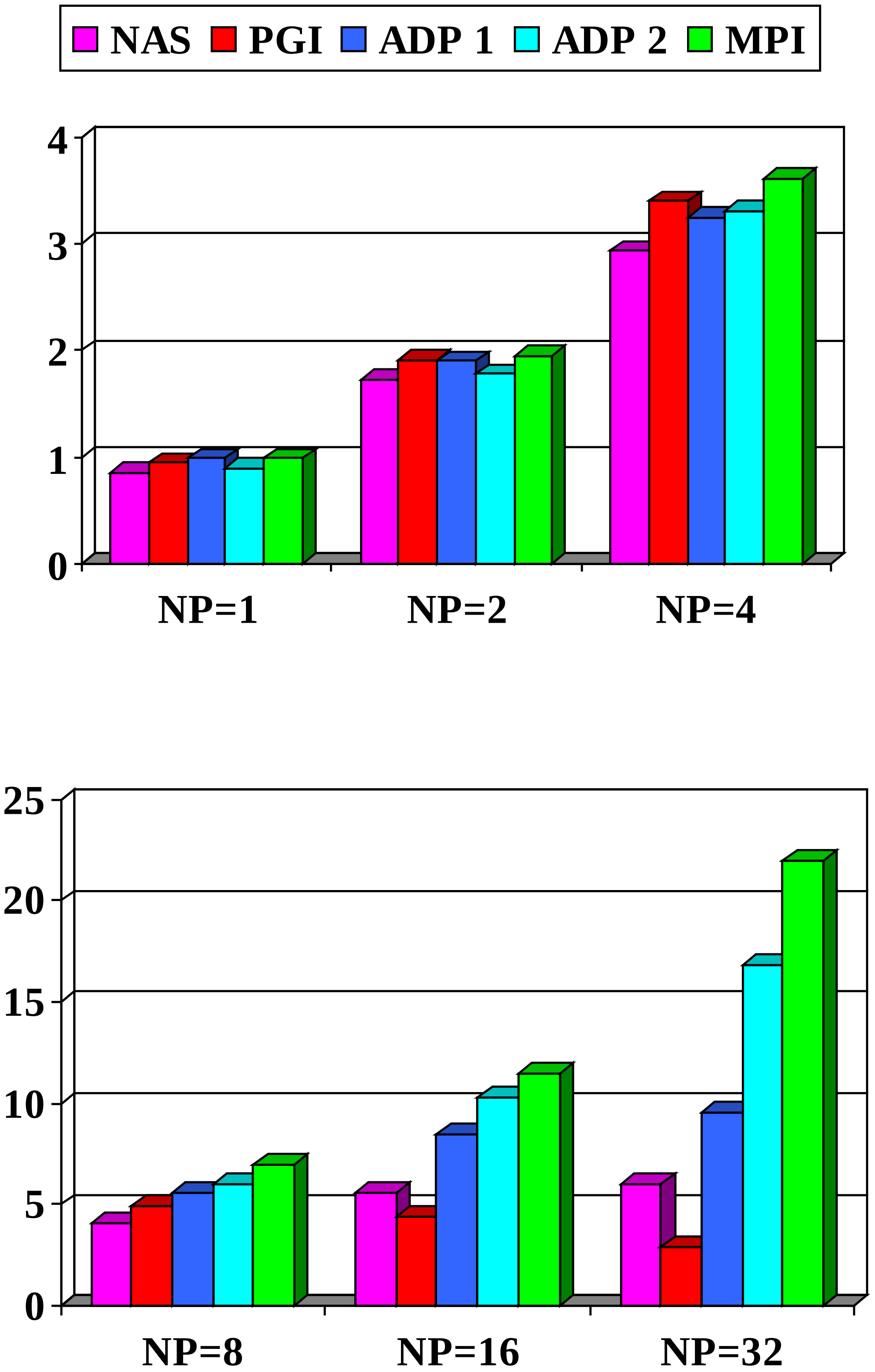
Execution Times: Medium

Industrial Test Case 3: Large

NAS:	NAS HPFPlus 2.01
PGI:	PG HPF
ADP 1:	ADAPTOR 5.1
ADP 2:	ADAPTOR 6.0a (traced)
MPI:	Aerolog V2P



Execution Times: LARGE



Speed-Ups: AEROLOG (HPF, MPI)

Effort Analysis

- **Most of the effort was spent on the reorganization of the data structures.**
- **Tuning was rather complex.**
- **Most effort could be avoided if HPF compilers improved (general block distributions, support of unstructured communication)**
- **Behavior of HPF programs difficult to understand**

Effort Analysis

Summary

- **Fortran 90/HPF version of AEROLOG available**
- **runs with at least 3 HPF compilers (portable)**
- **not fully portable (small differences in Syntax, HPF standard not very clear)**
- **no HPF / Fortran 90 / FORTRAN 77 overhead**
- **acceptable results for smaller number of processors**
- **does not scale well, will not replace MPI version**
- **wastes memory, will not replace F77 version**
- **porting, tuning effort higher than expected**

Conclusions

- **HPF is a useful paradigm for porting large FORTRAN 77 applications**
- **in the long run the better alternative**
- **but HPF compilers have to be improved**

Conclusions