Porting to HPF: Experiences with DBETSY3D within PHAROS

Dr. Thomas Brandes
Institute for Algorithms and
Scientific Computing (SCAI)
GMD, Germany
e-mail: brandes@gmd.de

Dr. Kadri Krause
debis Systemhaus
Leinfelden, Germany
e-mail: kadri@str.daimler-benz.com

HUG’98

Porto, June 1998

DBETSY3D

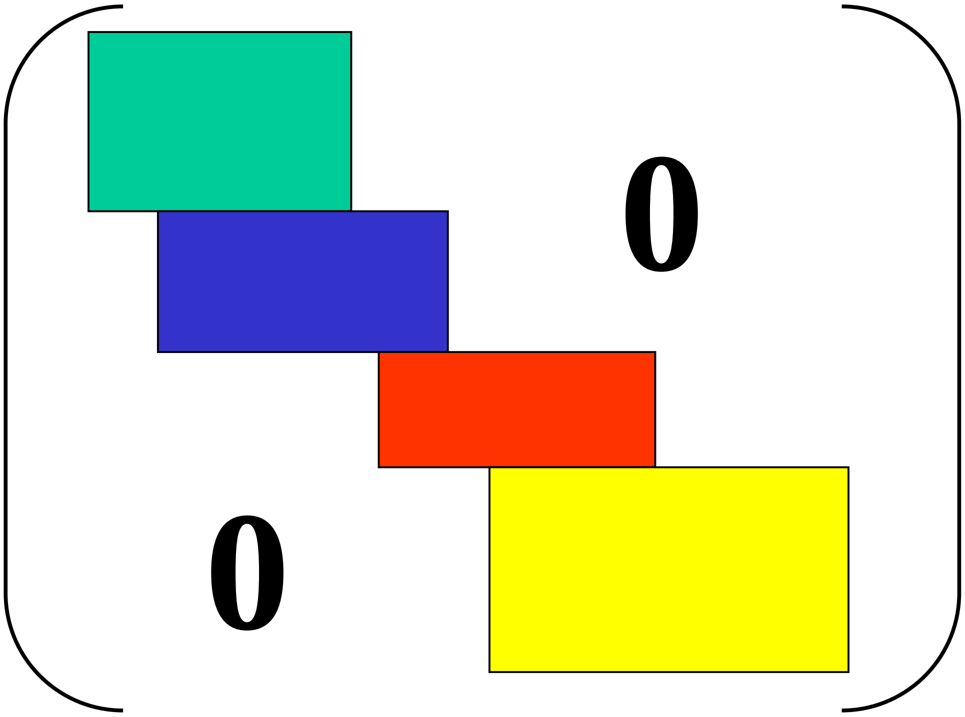
- **calculates stresses and deformations on the surfaces of objects**
- **Theory of linear electrostatics in three dimensional applications**
- **uses the Boundary Element Method (BEM)**
- **Advantage of BEM: Just the surface and not the volume must be modelled**
- **DBETSY3D started as the FORTRAN IV Betsy at the Forschungsvereinigung Verbrennungs-kraftmaschinen (Research Group Combustion Engines) in Frankfurt, Germany in 1977**
- **DBETSY3D is developed and owned by Mercedes Benz AG**
- **DBETSY3D is written in FORTRAN 77**
- **used extensively for calculating linear stresses**

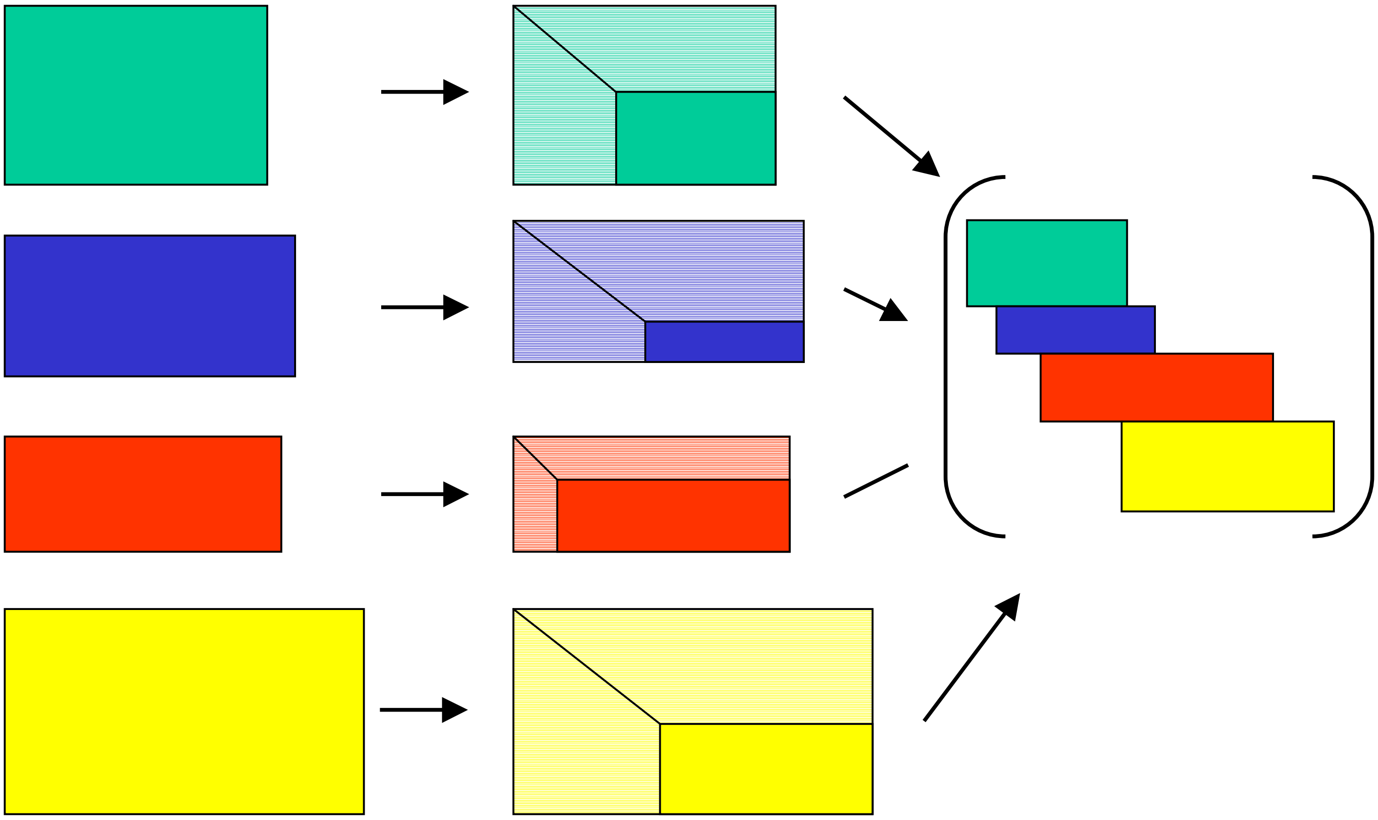
DBETSY3D

- DBETSY3D is composed of six programs
- the six programs run serially
- DBETSY5 requires the most CPU time

	KWREN	LEVER
DBETSY1	46.5 s	125.9 s
DBETSY2	2.4 s	4.2 s
DBETSY3	29.0 s	47.3 s
DBETSY4	249.4 s	703.6 s
DBETSY5	581.1 s	3761.2 s
DBETSY6	33.4 s	98.0 s
Total	941.8 s	4740.2 s

DBETSY5 is the Solver.





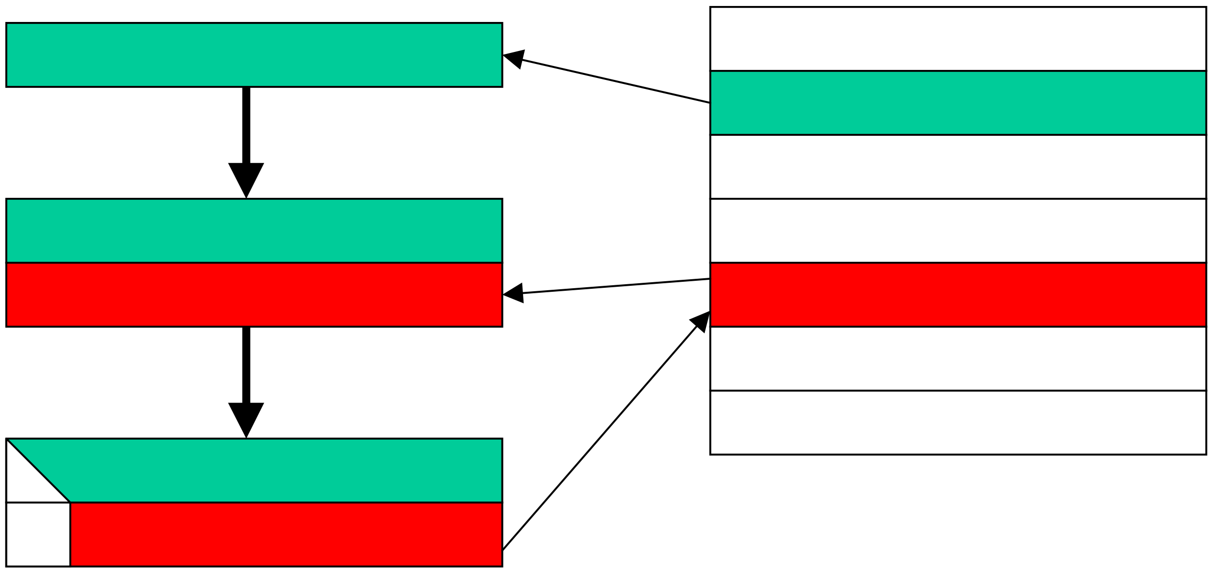
Residual Matrix Technique

Pseudo Code of DBETSY5

- read in problem sizes
- for each substructure
  - LU decompose it (forward substitution)
  - copy part of the LU decomposition into the residual matrix
- solve the residual matrix
- for each substructure
  - use solution of residual matrix
  - perform backward substitution
  - write substructure solution

Pseudo-Code of DBETSY5

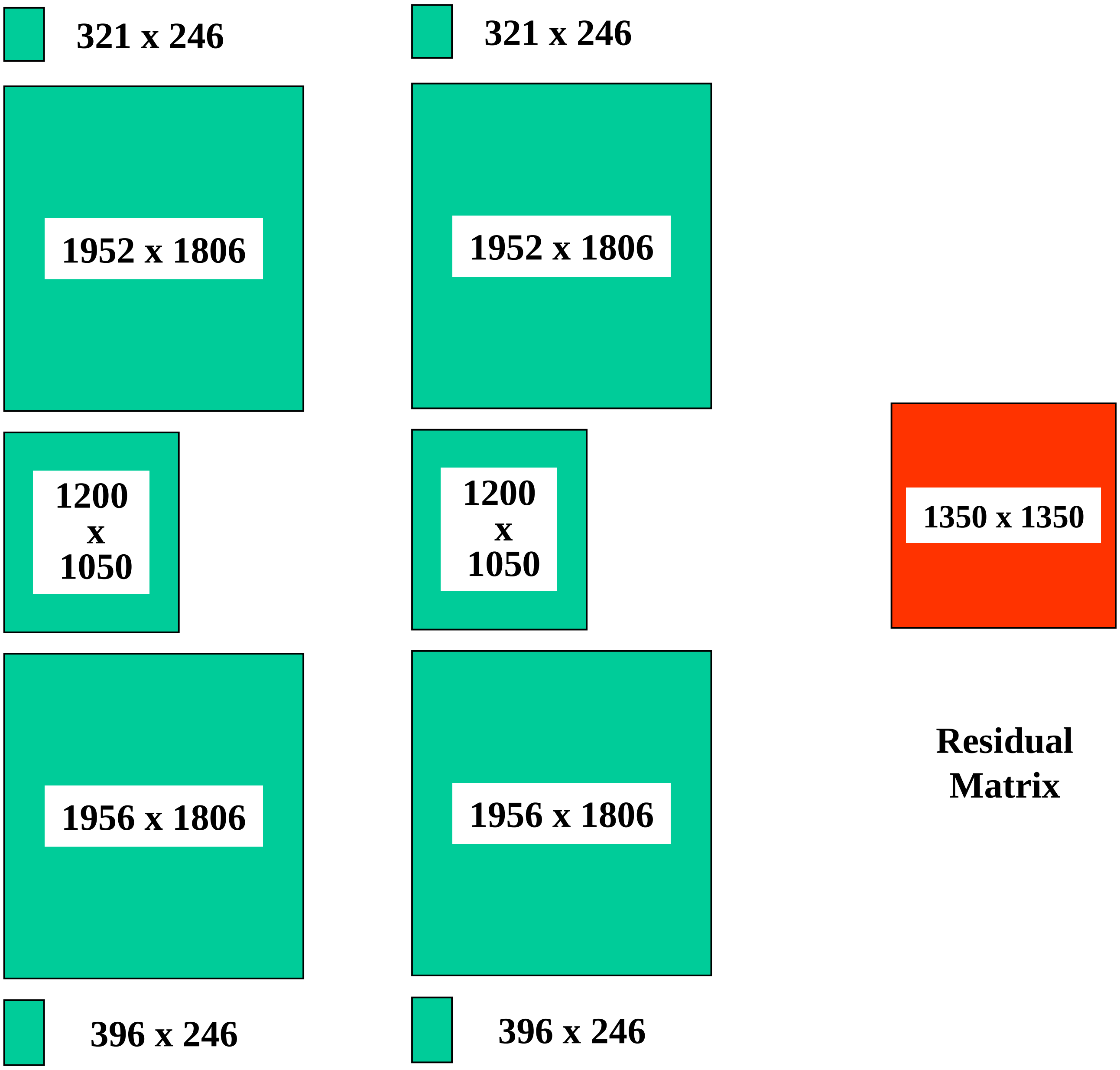
Block Solving Technique for Forward Substitution



in-core out-of-core

Only two blocks are kept in-core

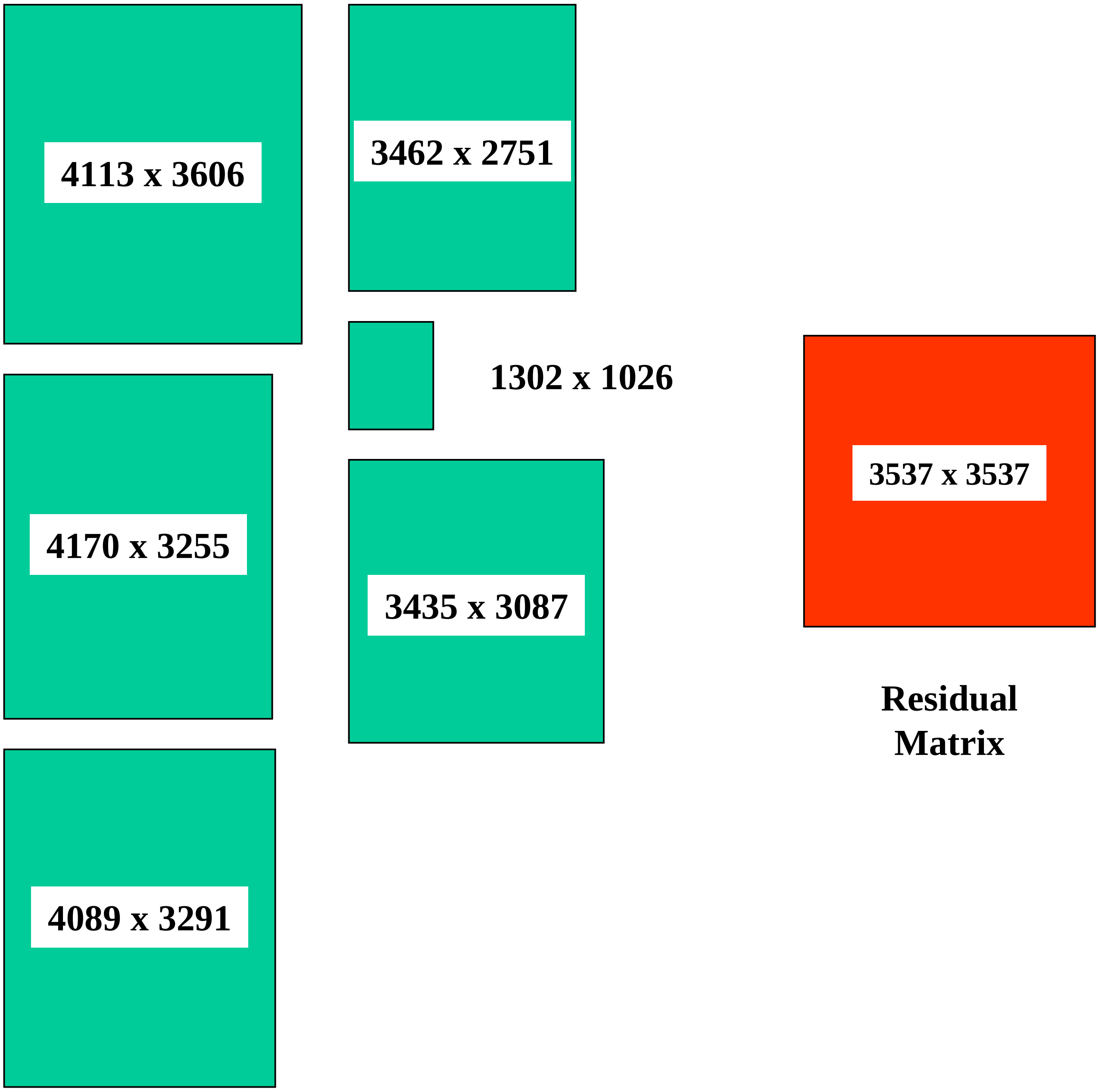
KWREN (crank shaft under gas pressure)



10 Subdomains

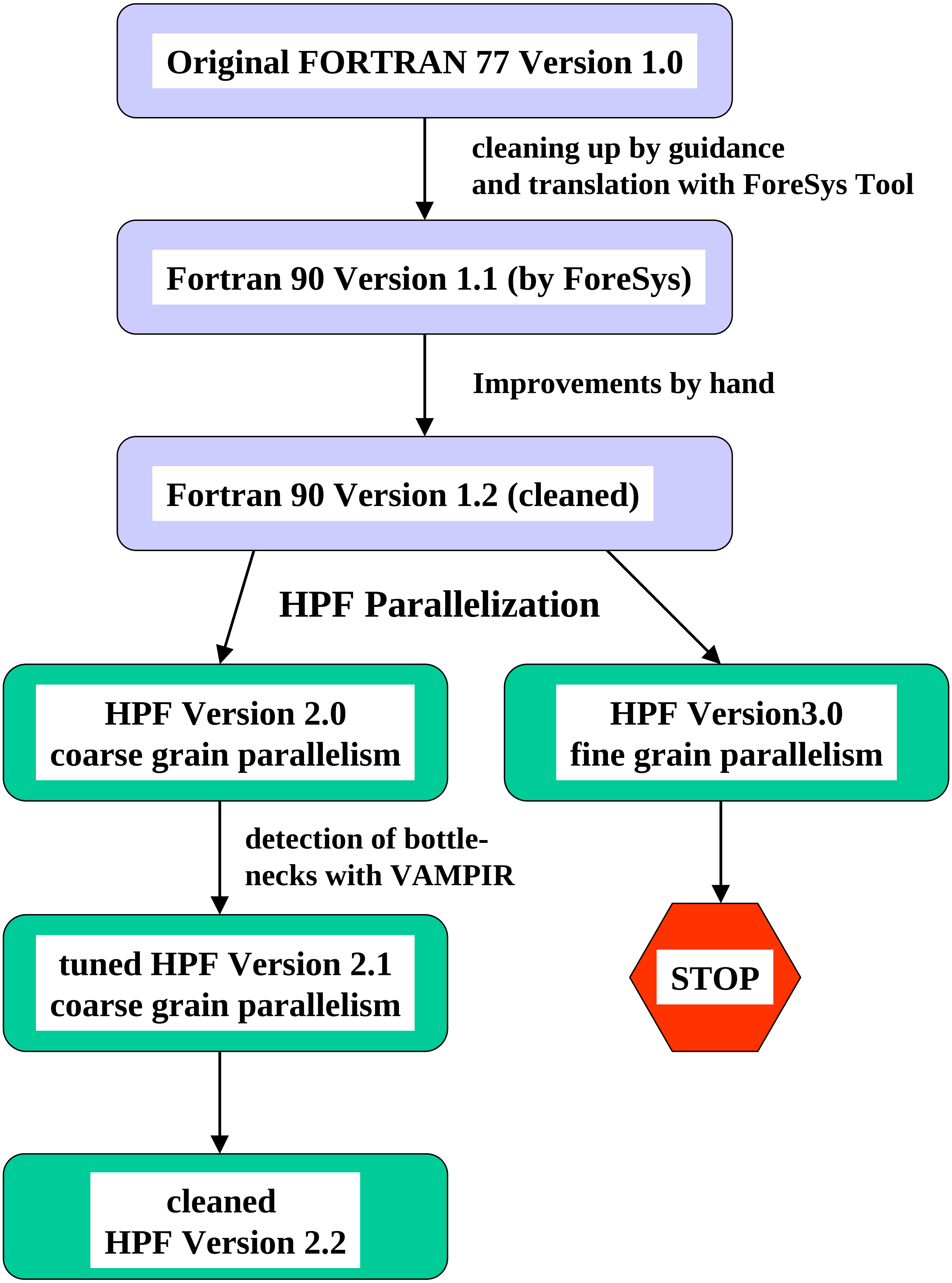
Testcase: KWREN

LEVER: wheel rocking lever



6 Subdomains

Testcase: LEVER



Versions of DBETSY5

Original FORTRAN 77 Version 1.0

↓ cleaning up by guidance
and translation with ForeSys Tool

Fortran 90 Version 1.1 (by ForeSys)

Initial cleaning up where ForeSys pointed out

- unnecessary arguments in argument list
- inconsistent names for variables in COMMON blocks
  - `common /CFIL/ I30, I3(MSS)`
  - `common /CFIL/ I3(MSS+1)`
- dead code
- unused variables

Automatic Translation to Fortran 90 (by ForeSys)

- new Fortran 90 syntax
- all variables are declared (`IMPLICIT none`)
- goto elimination (83 -> 21)
- intent for dummy arguments
- in/out usage of variables in `COMMON` blocks
- indentation

DBETSY5: Initial Fortran 90 Version

Fortran 90 Version 1.1 (by ForeSys)

↓ **Improvements by hand**

Fortran 90 Version 1.2 (cleaned)

Program Transformations (by hand):

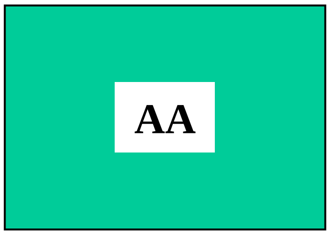
- reorganization of the workspace
- two-dimensional arrays instead of one-dimensional ones
- dynamic arrays
- array operations
- new subroutines
- removing sequence association
- removing loop dependencies
- reorganization of I/O handling
- many other improvements (better readability)

DBETSY5: Improvements on Fortran 90 version

```
COMMON // A(NSIZE)
```

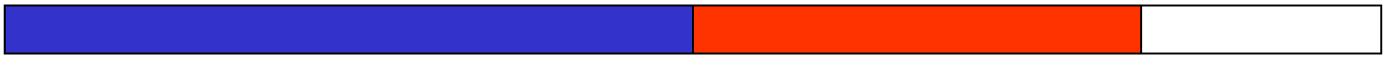
1. Use in Backward Substitution:

```
IADR = (J-1) * NSR
read (unit = ...) (A(IADR+K), K = 1, NSR)
```

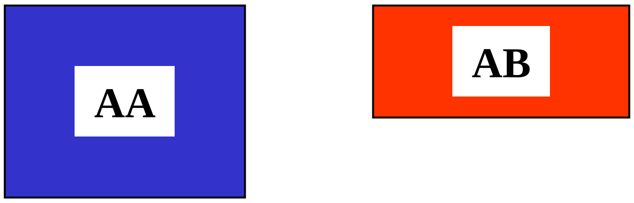



```
allocate (AA(NZB, NSR))
...
read (unit = ...) (AA(J,:))
```

2. Use in ‘Residual’



```
IADR1 = NZB * NSR
IADR2 = IADR1 + (II-1) * NZEIL
write (unit = ...) A(IADR2+K), K =1, NZEIL
```




```
allocate (AA(NZB,NSR),AB(NZB,NZEIL))
...
write (unit = ...) (AB (:,II))
```

Reorganization of WORKSPACE

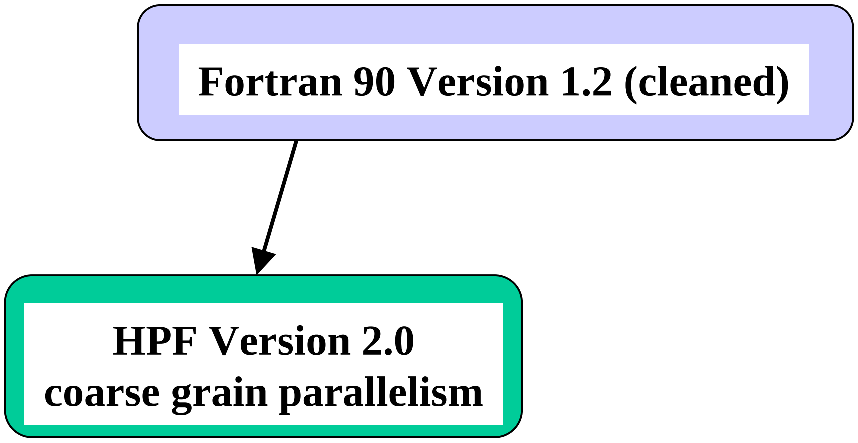
Code contained still loop dependencies:

```
      do I = 1, LUBC
         ...
         call RESBEL (..., NAKT, ...)
         ! NAKT is intent(inout) dummy argument
         ...
      end do
```

Calculation of NAKT forms a recurrence that could be solved:

```
!HPF$ INDEPENDENT, NEW(NAKT)
      do I = 1, LUBC
         ...
         NAKT = f(I)
         call RESBEL (..., NAKT, ...)
         ! NAKT is intent(in) dummy argument
         ...
      end do
```

Removing of Loop Dependencies

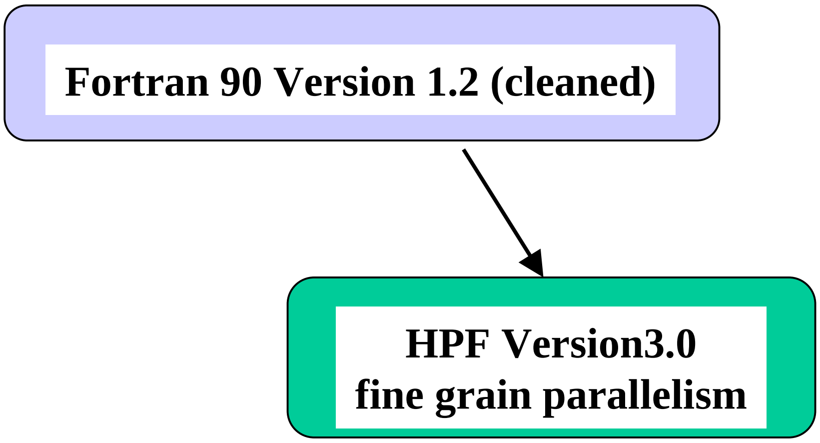


- **inserting some distributed arrays**
- **error handling**
- **inserting `INDEPENDENT` directive for loops**

```
      integer, dimension (LUBC) ::IERROR
!hpf$ distribute (BLOCK) :: IERROR
...
!hpf$ independent
      do LUB=1,LUBC
         call FORWARD (IERROR(LUB), LUB, NZEIL, ...)
      end do
```

- **FORWARD is not a pure Routine (I/O, but cheat the compiler)**
- **NAS HPFPlus: solution with `EXTRINSIC (HPF_SERIAL)`**
- **but not portable: use of COMMON data**
- **Concurrent WRITE to file (for residual matrix) must be allowed**

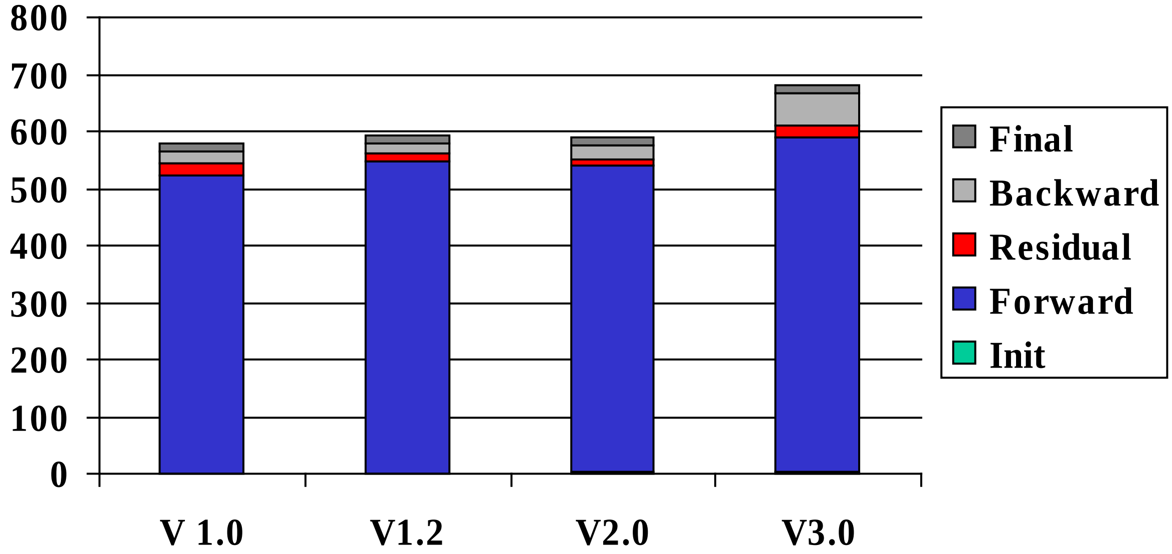
HPF Parallelization: Coarse Grain Parallelism



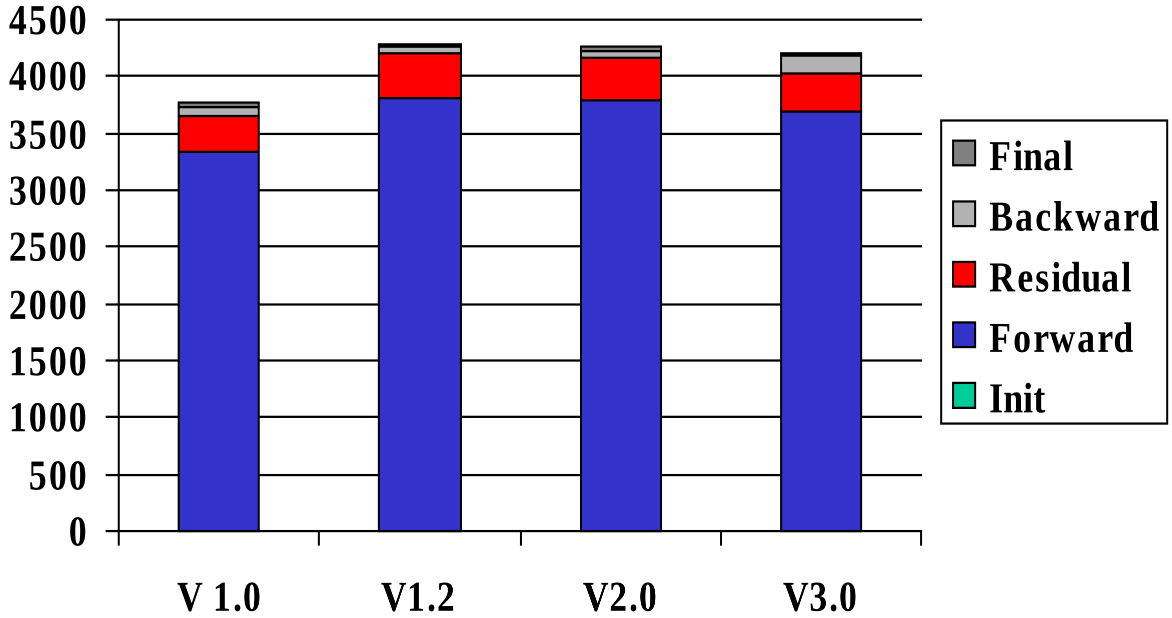
- most useful for forward substitution
- most work already done in step before (cleaning)
- distribution of local arrays (BLOCK or CYCLIC(1))
- distribution directives for dummy arrays
- parallelism by INDEPENDENT, NEW directive

HPF Parallelization: Fine Grain Parallelism

KWREN

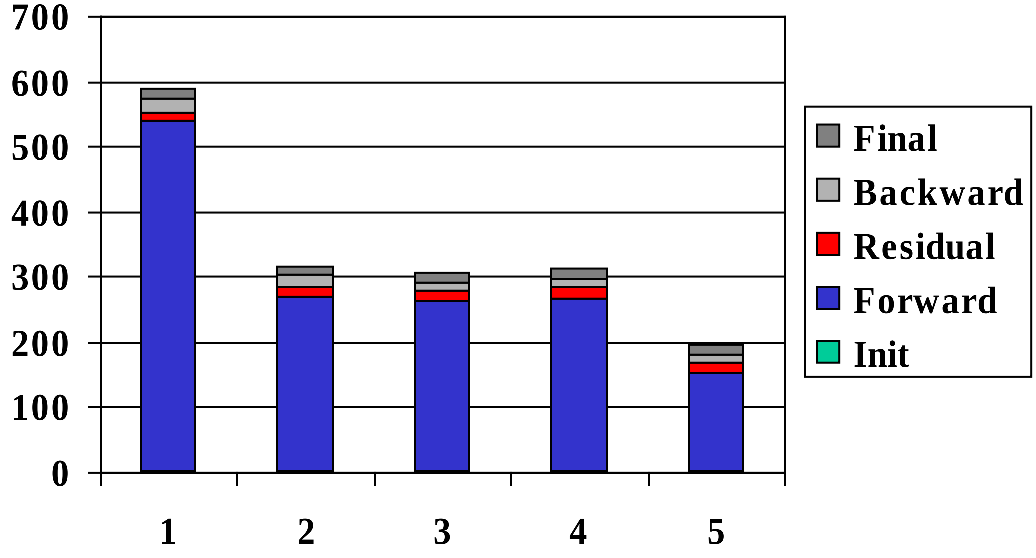


LEVER



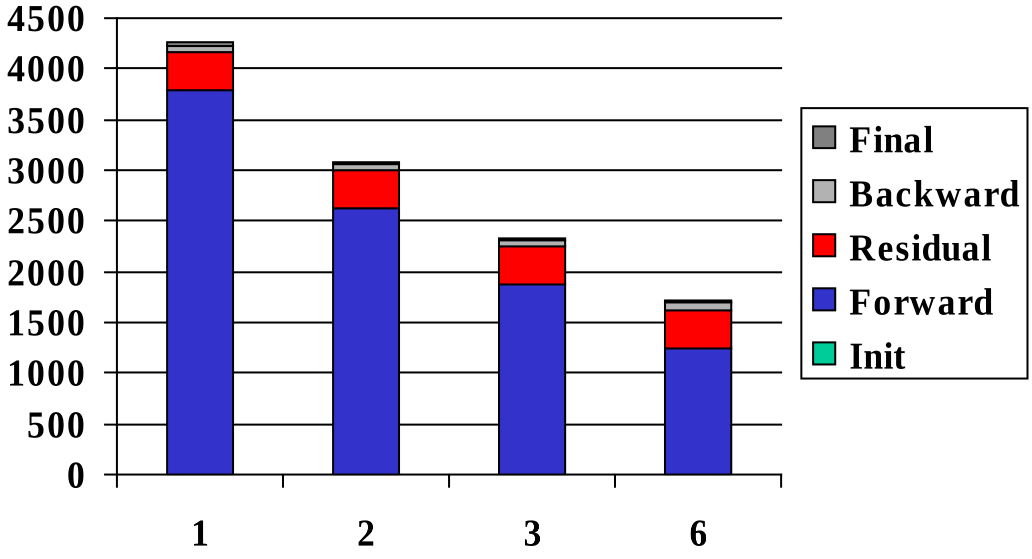
Serial Execution Times, IBM SP2, thin node

KWREN



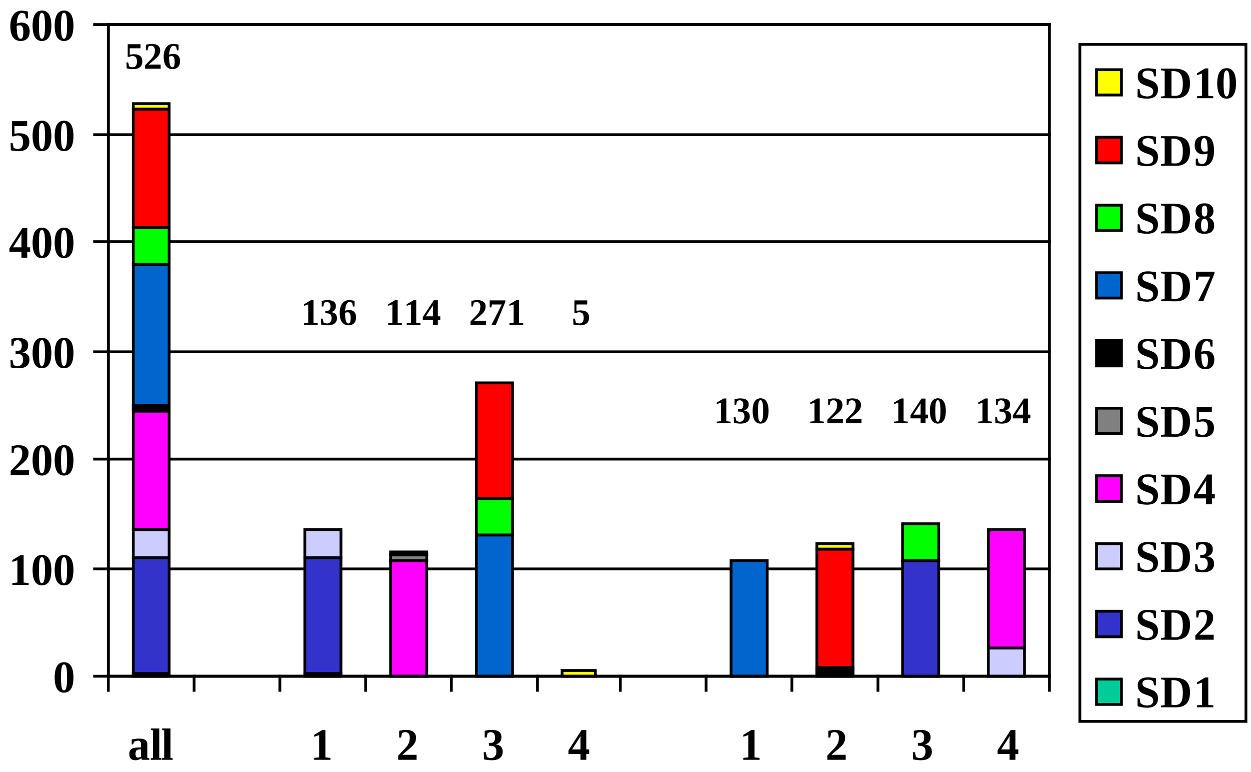
(real) Speed-Up: 0.99 1.82 1.88 1.85 2.93

LEVER

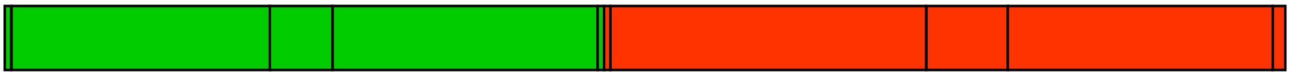


(real) Speed-Up: 0.88 1.26 1.61 2.19

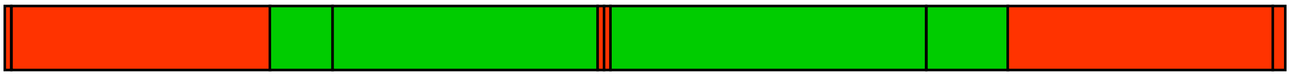
Parallel Times of V2.0: HPF Coarse Grain



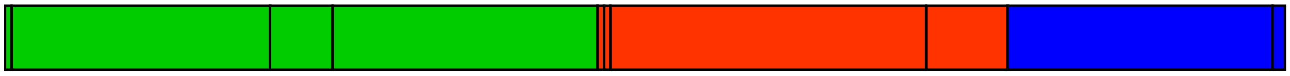
NP = 2, P1: 247, P2: 279



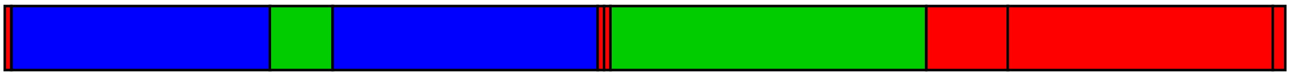
NP = 2, P1: 264, P2: 262



NP = 3, P1:243, P2: 170, P3: 113



NP = 3, P1: 157, P2: 155, P3: 214



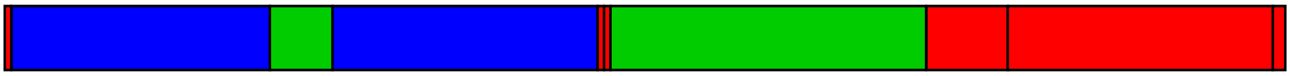
Load Balancing: KWREN on 4 Processors

```
      integer, dimension (LUBC) ::IERROR
      real, dimension (LUBC) :: LOAD    ! Work
      integer, dimension (LUBC) :: MAP   ! mapping
!hpf$ distribute (BLOCK) :: IERROR
!hpf$ dynamic IERROR
      ...
      call GETLOAD (LOAD, LUBC)
      call MAPPING (LOAD, MAP, LUBC)
!hpf$ redistribute IERROR (INDIRECT(MAP))
!hpf$ independent, on home (IERROR(LUB))
      do LUB=1,LUBC
         call FORWARD (IERROR(LUB), LUB, NZEIL, ...)
      end do

      subroutine GETLOAD (LOAD, LUBC)
      integer, intent(in) :: LUBC
      real, dimension(LUBC), intent(out) :: LOAD
      ...  ! Estimate time for each subdomain
      end subroutine GETLOAD

      subroutine MAPPING (LOAD, MAP, LUBC)
      integer, intent(in) :: LUBC
      real, dimension(LUBC), intent(in) :: LOAD
      integer, dimension(LUBC), intent(out) :: MAP
      ... ! Greedy algorithm for the mapping
      end subroutine MAPPING
```

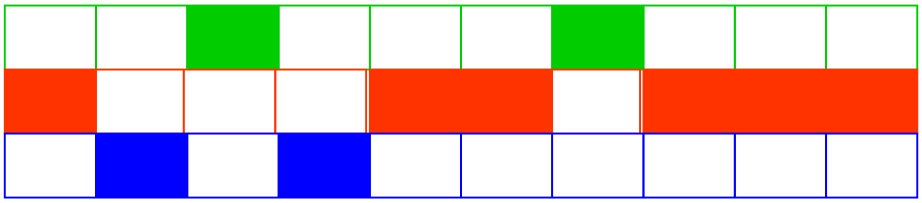
Realization of Load Balancing

```
      logical, dimension(NP,LUBC) :: DOIT
!     DOIT (P,LUB) : processor P works on SD(LUB)
      ...
      integer, dimension (NP,LUBC) :: IERROR
!hpf$ distribute(block,*) :: IERROR
      DOIT = .false.
      do LUB = 1, LUBC
         DOIT (MAP(LUB)) = .true.
      end do

!hpf$ independent
      do P=1, NP
        do LUB=1,LUBC
           if (DOIT(P,LUB)) then
             call FORWARD (IERROR(P,LUB), LUB, NZEIL, ...)
           end if
        end
      end do
```

Example: KWREN, NP=3

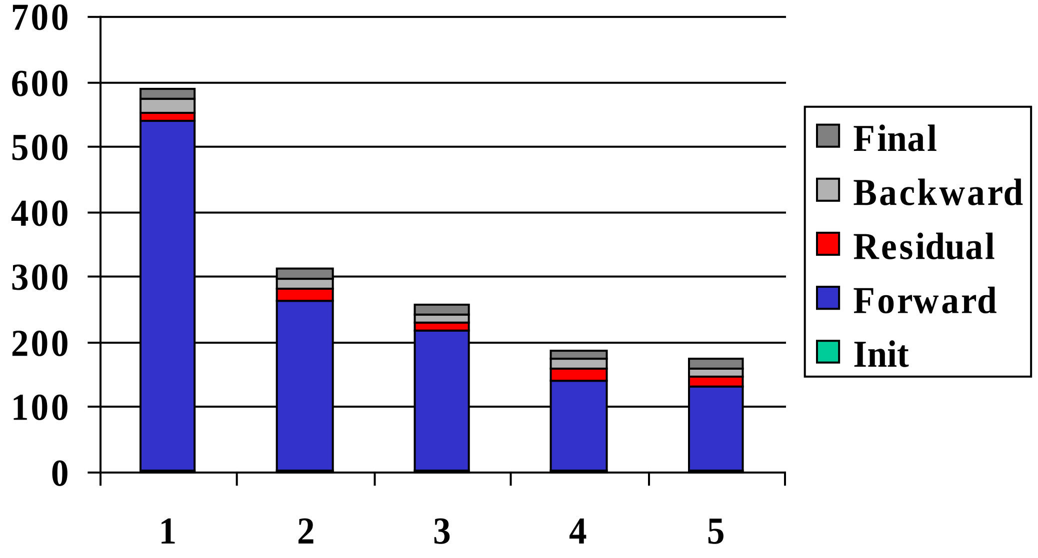


DOIT (3,10)



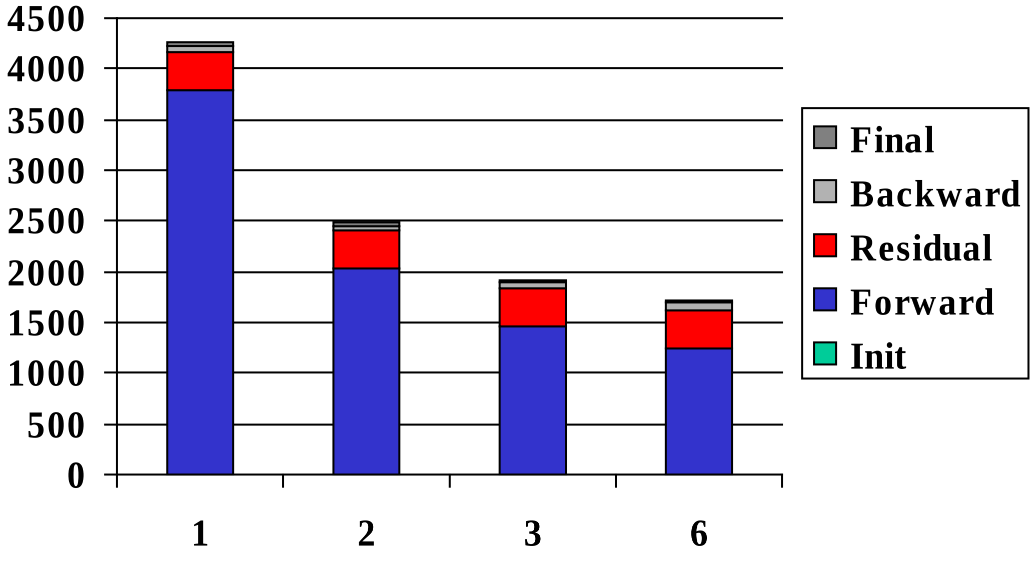
Workaround: Load Balancing

KWREN



(real) Speed-Up:	0.99	1.85	2.23	3.06	3.14
before:	0.99	1.82	1.88	1.85	2.93

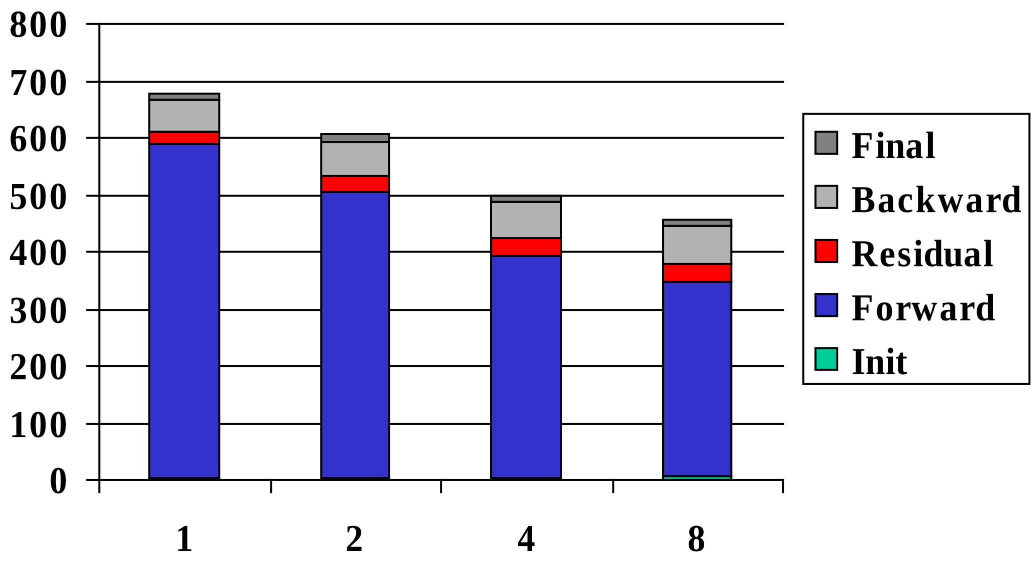
LEVER



(real) Speed-Up:	0.88	1.58	1.95	2.19
before:	0.88	1.26	1.61	2.19

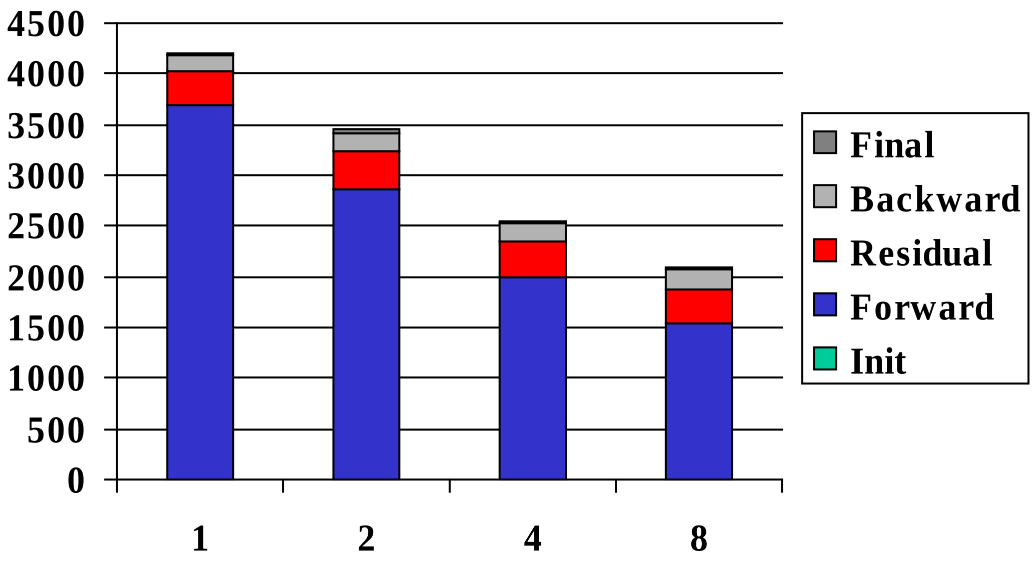
Parallel Times of V2.1: Load Balancing

KWREN



(real) Speed-Up: 0.86 0.96 1.16 1.26

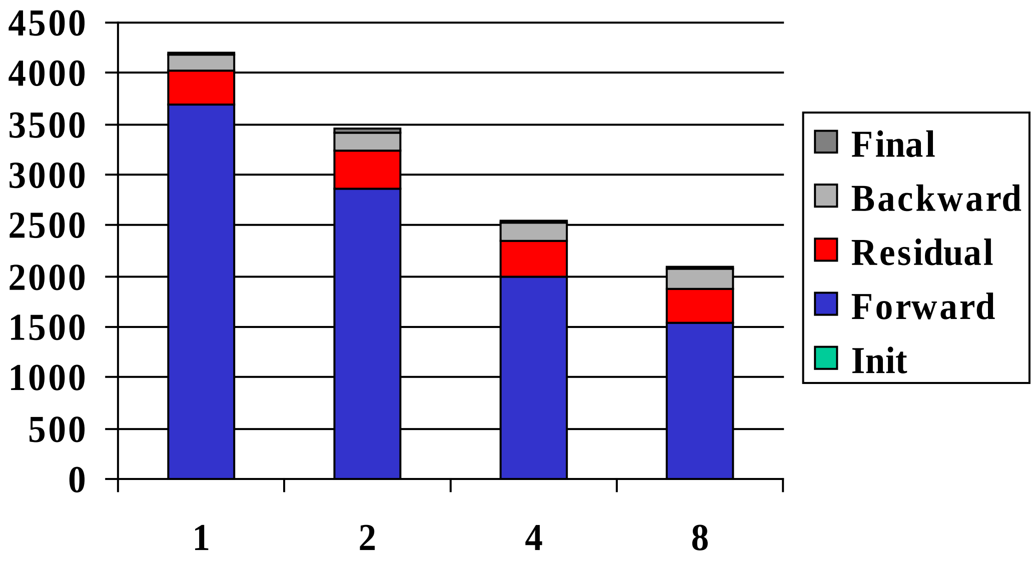
LEVER



(real) Speed-Up: 0.89 1.09 1.48 1.75

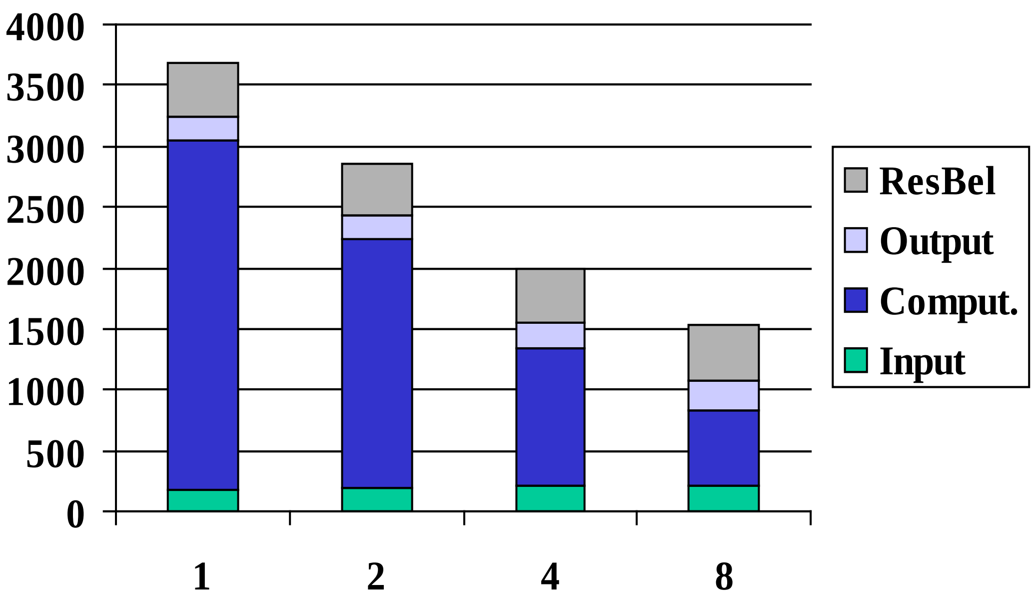
Parallel Times of V3.0: Fine Grain Parallelism

LEVER



(real) Speed-Up: 0.89 1.09 1.48 1.75

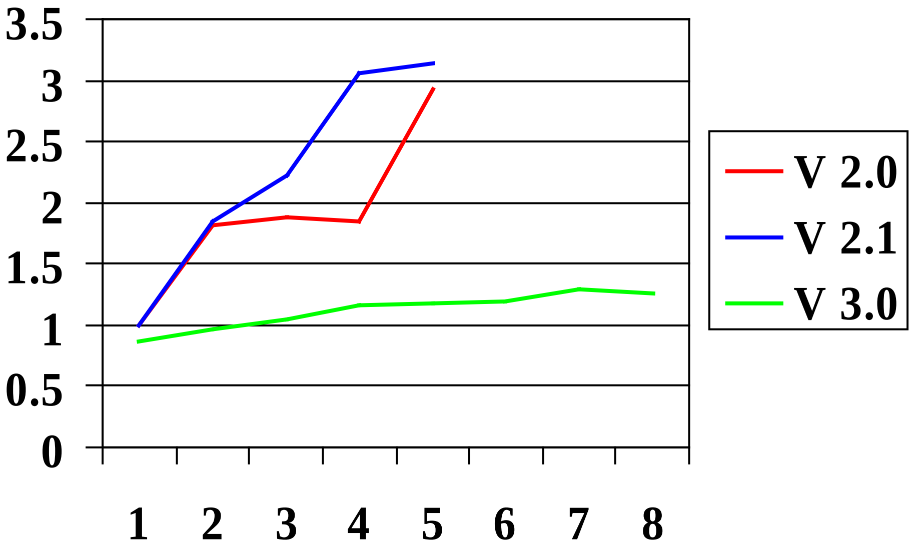
Forward Substitution (only)



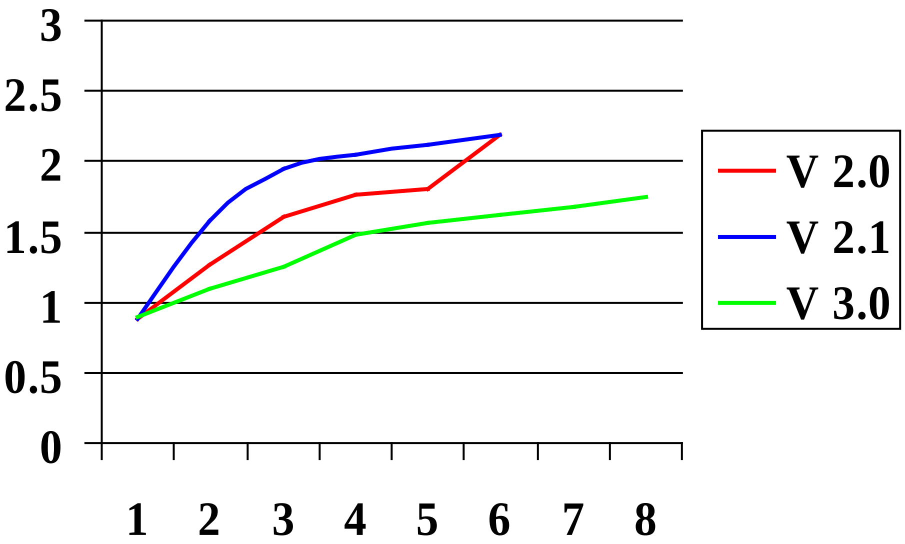
(real) Speed-Up: 0.90 1.16 1.67 2.17

V3.0: Detailed Results

KWREN



LEVER



DBETSY5: Real Speed-Ups

ForeSys

- very useful
- easy to apply
- could not be used for the Fortran 90 code

NAS HPFPlus

- Fast and good support
- made good progress during project lifetime
- required features were available
- no support of `INDEPENDENT, NEW`
- NAS HPF Debugger not really used

Vampir

- Good graphical user interface, high functionality
- documentation is good
- problems with large trace files
- no information about global communication
- very good interaction VAMPIR - HPFPlus
- no real use for the unexperienced user
- high overhead to use for experienced user

Effort Analysis

- **Most of the effort was spent on the conversion to Fortran 90. We do have much cleaner and more readable code.**
- **The conversion from Fortran 90 to HPF was much faster and much easier than writing MPI code.**
- **HPF code is expected to be much easier to maintain than MPI code.**

Effort Analysis

Conclusions

- Fortran 90/HPF version of DBETSY5 available
- well-structured code now (better maintainability)
- no Fortran 90 or HPF overhead introduced
- coarse-grain parallelism scales better
- load balancing absolutely necessary (currently with a workaround)
- same parallelization strategy as MPI version, nearly the same performance
- fine-grain parallelism only with parallel I/O (but not standardized at this time)

Conclusions