Application of High Performance FORTRAN

Edward Bogucz Geoffrey Fox
Tom Haupt Kenneth Hawick
Kevin Roe

CRPC-TR95619
July 1995

Center for Research on Parallel Computation
Rice University
6100 South Main Street
CRPC - MS 41
Houston, TX 77005

Also available as *SCCS-727* from the Northeast Parallel Architectures Center at Syracuse University.

Application of High Performance Fortran

Geoffrey Fox, Ken Hawick, Tom Haupt

Edward Bogucz and Kevin Roe

Northeast Parallel Architectures Center

Syracuse University

111 College Place

Syracuse NY 13244-4100

http://www.npac.syr.edu/

NPAC Technical Note SCCS-727

Contents:

- Brief HPF Overview
- Issues for Solver algorithms in HPF
- ADI Algorithm - conflict in optimal data decomposition
- Panel Method and Full Matrix (Direct) Algorithms
- Sparse Matrices in HPF - Conjugate Gradient Algorithm
- Selected NPAC Projects
- Conclusions on HPF applicability for CFD problems
- HPF Applications Web Package

HPF Overview:

- High Performance Fortran (HPF): adds to Fortran 90 by allowing various directives to specify data decompositions;
- Explicit parallel constructs like FORALL;
- Additional parallel intrinsics functions - optimised on particular architecture.
- Many Details given elsewhere (see WebPackage including tutorial or HPF Forum page)

Solver Issues for HPF(1):

- HPF's data-parallel model excellent for mesh/stencil methods for initial value problems
- regular mesh boundary value problem still good although some data-parallelism wasted during time to propagate values in from boundary.
- many intrinsics functions do BLAS like operations on dense vectors and arrays.
- line solvers like ADI can work well iff implementation has good communications support for run time library.
- as for all parallel solver algorithms, tradeoff of memory-efficiency and compute-efficiency occurs.

Solver Issues for HPF(2):

- direct (full matrix) methods can be expressed in HPF but are not as efficient as the task parallelism that pure message passing can achieve (eg ScaLAPACK could be a good run time library module)
- unstructured sparse systems can be implemented using INDEPENDENT but are inefficient without new features planned for HPF-2 (private data for processors)
- structured sparse sytems can be efficient in storage space (store diagnonals as vectors etc, providing iterative method used instead of direct method and can avoid fill-in problems.)

Some Case Studies:

- Alternate Direction Implicit Problem Formulation;
- Panel Method (dense linear algebra);
- Iterative methods (conjugate gradient).

Alternate Direction Implicit Problem Formulation:

- Consider Equation:

$$\nabla^2\psi = f \quad on \quad \Omega = (0,1)\times(0,1), \tag{1}$$
$$\psi = \psi_g \quad on \quad \partial\Omega, \tag{2}$$

- Dirichlet boundary conditions on ψ.
- $N_x + 1$ intervals in x and $N_y + 1$. intervals in y

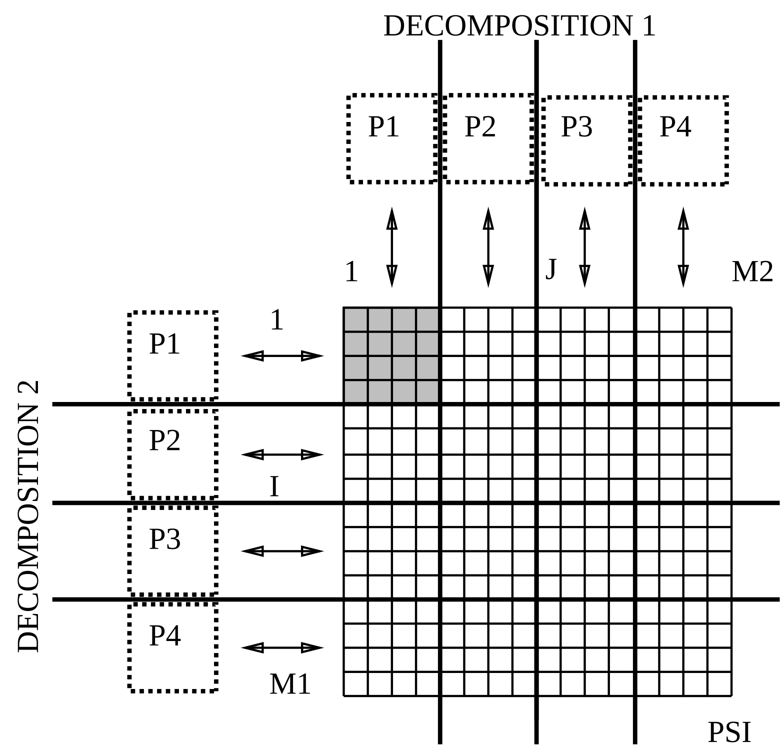
```
     PROGRAM ADI_SEQUENTIAL
...  declarations, interfaces and initializations ...
     DO ITER = 1,ITERMAX      ! begin iteration loop
       DO I = 1,NX            ! x sweep
         C(1,1:NY) = DY2INV
         C(2,1:NY) = -2.0_FPNUM*(DX2INV+DY2INV)
         C(3,1:NY) = DY2INV
         C(4,1:NY) = F(I,1:NY)-DX2INV*(PSI(I+1,1:NY)+PSI(I-1,1:NY))
         CALL THOMAS(NY,C,BCBOT,BCTOP,PSI(I,0:NY+1))
       END DO
       DO J = 1,NY !y sweep
         C(1,1:NX) = DX2INV
         C(2,1:NX) = -2.0_FPNUM*(DX2INV+DY2INV)
         C(3,1:NX) = DX2INV
         C(4,1:NX) = F(1:NX,J)-DY2INV*(PSI(1:NX,J+1)+PSI(1:NX,J-1))
         CALL THOMAS(NX,C,BCLFT,BCRHT,PSI(0:NX+1,J))
       END DO
...  check convergence ...
       END DO
     END
```

Thomas algorithm for ADI code:

```
      SUBROUTINE THOMAS (NK,C,ZO,ZN,Z)
...   declarations ...
      D(1,0) = 0.0
      D(2,0) = ZO
      DO K=1,NK
        D(1,K) = -C(1,K)/(C(2,K) + C(3,K)*D(1,K-1))
        D(2,K) = (C(4,K)-C(3,K)*D(2,K-1))/ &
                 (C(2,K) + C(3,K)*D(1,K-1))
      END DO
      Z(NK+1) = ZN
      DO K=NK,1,-1
        Z(K) = D(1,K)*Z(K+1) + D(2,K)
      END DO
      Z(0) = ZO
      RETURN
      END
```

Modified x- and y-sweeps for data parallel execution:

```
      DO I = 1,NX      !x sweep
      C(1,I,1:NY) = DY2INV
      C(2,I,1:NY) = -2.0_FPNUM*(DX2INV+DY2INV)
      C(3,I,1:NY) = DY2INV
      C(4,I,1:NY) = F(I,1:NY) - &
                    DX2INV*(PSI(I+1,1:NY)+PSI(I-1,1:NY))
      END DO
...   solve a set of tridiagonal system of equations
      DO J = 1,NY        !y sweep
      C(1,J,1:NX) = DX2INV
      C(2,J,1:NX) = -2.0_FPNUM*(DX2INV+DY2INV)
      C(3,J,1:NX) = DX2INV
      C(4,J,1:NX) = F(1:NX,J) - &
                    DY2INV*(PSI(1:NX,J+1)+PSI(1:NX,J-1))
      END DO
...   solve a set of tridiagonal system of equations
```

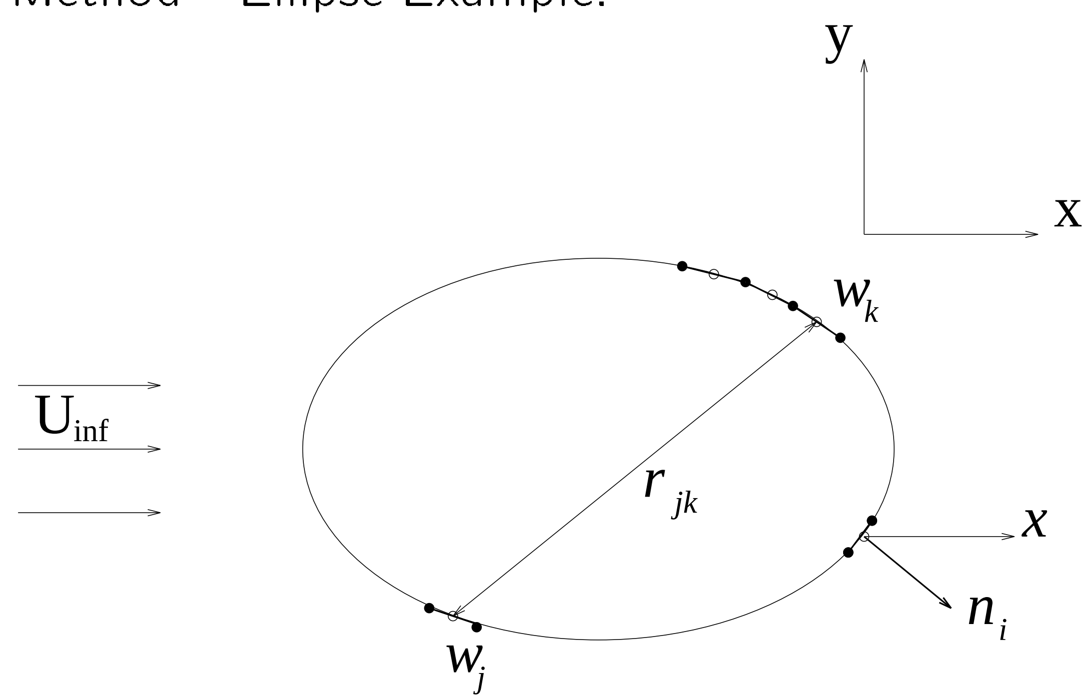


Optimal Data Decompositions for Matrix equations in ADI Solver (2D problem)

ADI Conclusions:

- ADI can be neatly expressed in HPF.
- many of BLAS type operations are in-line equivalents of Fortran 90 intrinsics. (eg SAXPY is one line of code, MATMUL intrinsic replaces all dense array multiplications.)
- REDISTRIBUTE is used instead of TRANSPOSE for 3d and higher dimensionality problems
- Performance depends on how well intrinsics and run time library is implemented.

Panel Method - Ellipse Example:



Ellipse in uniform incident flow U_{inf}, showing k'th panel of source strength $w_k \int ds_k$ at r_k.

Panel Method Formulation: Body in a uniform velocity U_{inf}, distribution of N source panels produces potential:

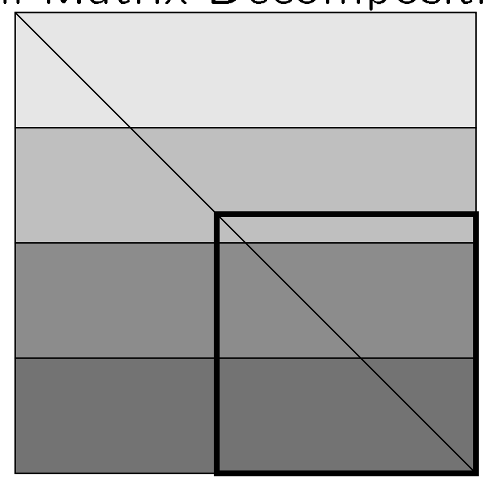
$$\Phi(\vec{r}_k) = U_0 x_k + \frac{1}{2\pi} \sum_{j=1}^{N} w_j \int \ln |\vec{r}|_{k,j} ds_j, \tag{3}$$

$\vec{r}_k = (x_k, y_k)$ is position of each panel's control point, $|\vec{r}|_{k,j}$ is distance between two panels, and $w_k \int ds_k$ is the source strength of the k'th panel. Source densities determined from boundary condition of zero normal flow through body surface: $v_n = \frac{\partial \Phi}{\partial n_k} = 0$ generates system of linear equations $A \cdot \vec{w} = \vec{b}$ with each component of A:

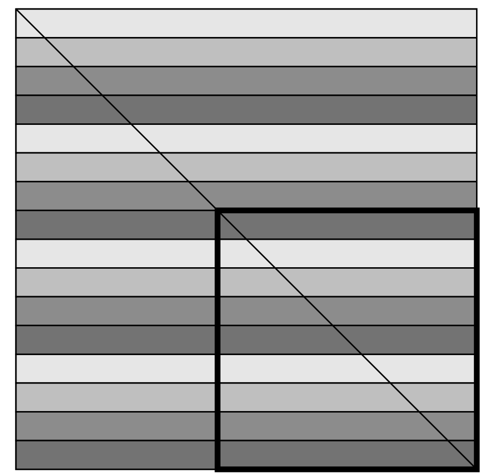
$$A_{k,j} = \frac{\delta_{k,j}}{2} + \frac{1}{2\pi} \int \frac{\partial}{\partial n_k} (\ln r_{k,j}) ds_j, \tag{4}$$

and RHS vector is $b_k = U_0 \sin \alpha_k$, where α_k is angle between panel and x-axis. Once vector of source densities determined, velocity field obtained from potential.

Full Matrix Decomposition:



BLOCK

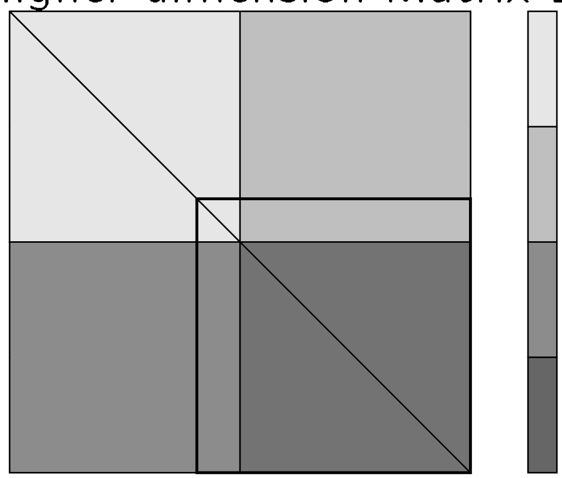


CYCLIC

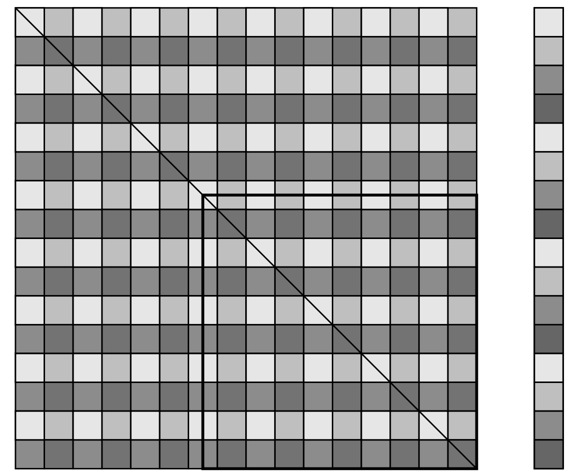
Row distribution across (4) processors' memories. (Column is similar).

Row distribution requires a distributed global test for the pivot, whereas for column this is poorly balanced - but requires no communication. Row requires broadcast of partial matrix row, column requires broadcast of multiplication factor.

Higher dimension Matrix Distribution



BLOCK



CYCLIC

Cyclic row distribution provides best decomposition for both matrix and RHS vector operations.

Column distributions are poor since a single processor is required to work on entire RHS at all stages of back -substitution. Mixing distributions is also possible but REDISTRIBUTE may be costly.

Full Matrix Conclusions:

- HPF allows problem to be expressed quite neatly
- as important to remove serial code as to add parallel code sections
- Once code is in HPF, it allows different decompositions to be tuned with only minor code changes - eg (BLOCK,*) easily changed to (*,CYCLIC)
- which of the decompositions performs best depends on problems sizes and characteristics of particular computer system (how well TRANSPOSE and REDISTRIBUTE are implemented)
- present implementations cannot outperform message passing for this sort of application (for example ScaLAPACK library), although in future ScaLAPACK may be part of run time library invoked by the compiler on programmers behalf.

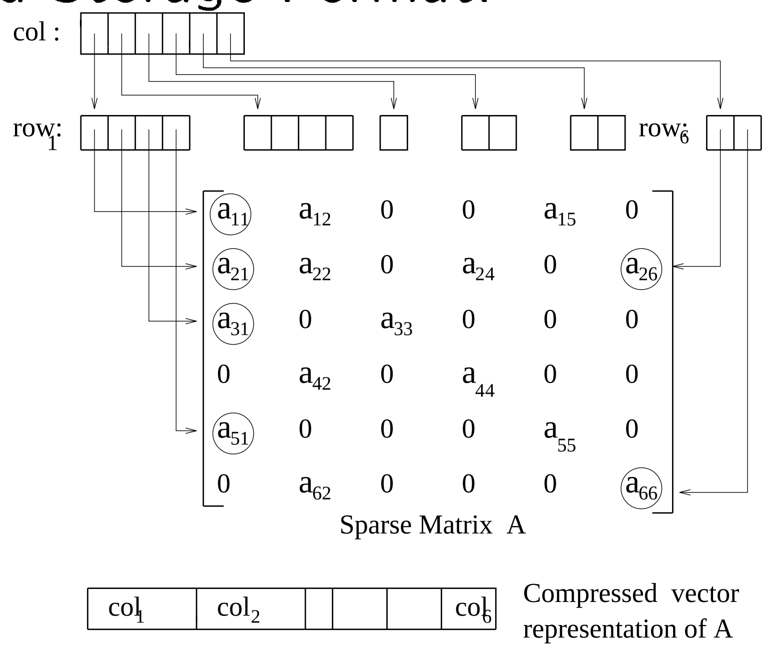
Sparse Methods - Conjugate Gradient example. The non-preconditioned CG algorithm is summarised as:

$\vec{p} = \vec{r} = \vec{b}$; $\vec{x} = 0$; $\vec{q} = A\vec{p}$
$\rho = \vec{r} \cdot \vec{r}$; $\alpha = \rho/(\vec{p} \cdot \vec{q}$
$\vec{x} = \vec{x} + \alpha\vec{p}$; $\vec{r} = \vec{r} - \alpha\vec{q}$
DO $k = 2$, Niter
    $\rho_0 = \rho$; $\rho = \vec{r} \cdot \vec{r}$; $\beta = \rho/\rho_0$
    $\vec{p} = \vec{r} + \beta\vec{p}$; $\vec{q} = A \cdot \vec{p}$
    $\alpha = \rho/\vec{p} \cdot \vec{q}$
    $\vec{x} = \vec{x} + \alpha\vec{p}$; $\vec{r} = \vec{r} - \alpha\vec{q}$
    IF(stop_criterion)exit
ENDDO

for the initial “guessed” solution vector $\vec{x}^0 = 0$.

Implementation of this algorithm requires storage for four vectors:, $\vec{x}$, $\vec{r}$, $\vec{p}$ and $\vec{q}$ as well as the matrix A and working scalars α and β.

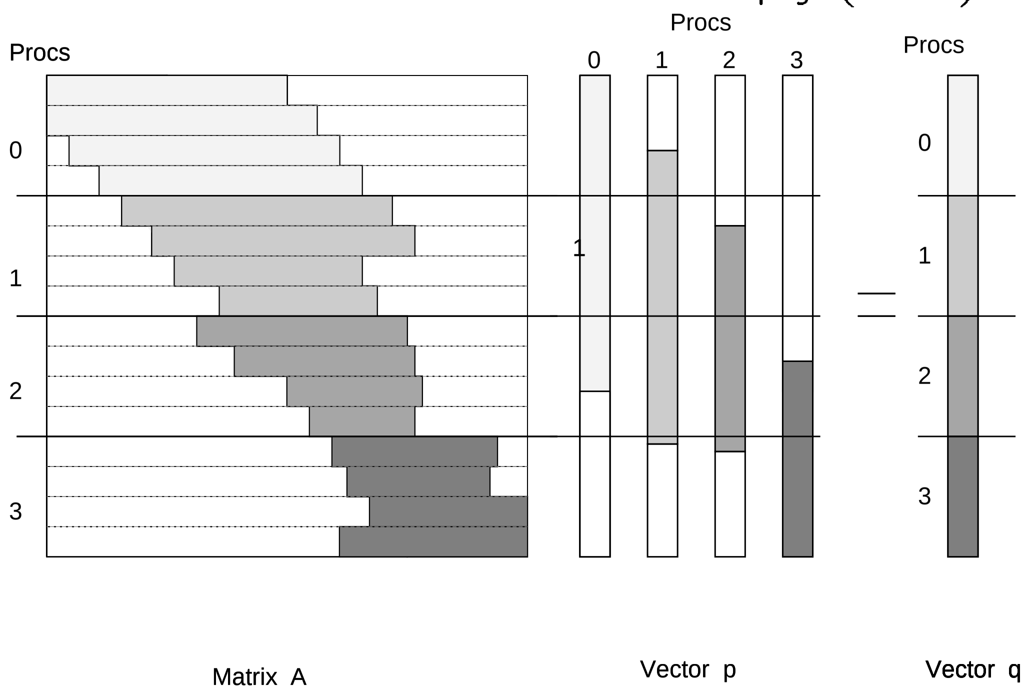
Compressed Storage Format:



Compressed Sparse Column(CSC) representation of sparse matrix A.

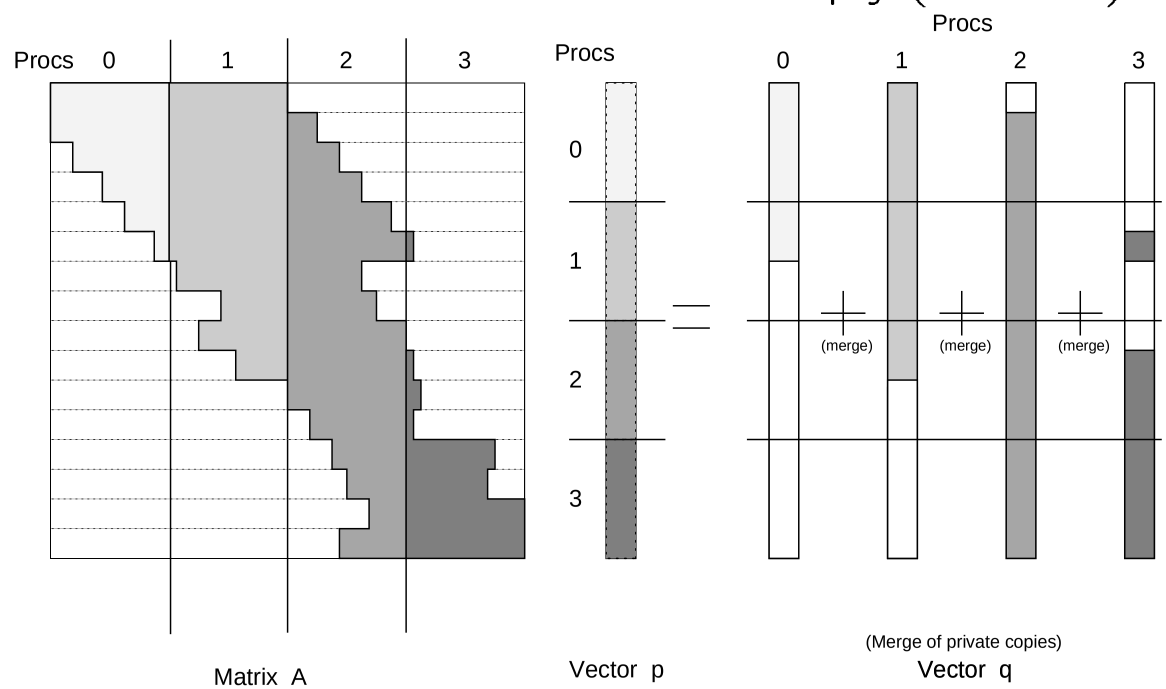
`A(nz)` contains the nonzero elements stored in the order of their columns from 1 to n; **`row(nz)`** stores the row numbers of each nonzero element; jth entry of **`col(n+1)`** points to the first entry of the j'th column in **`A`** and **`row`**.

Communications for Matrix Vector Multiply (Row)



Communication requirements of (sparse)Matrix vector multiplication where A is distributed in a (BLOCK, *) fashion.

Communications for Matrix Vector Multiply (Column):



Communication requirements of (sparse)Matrix vector multiplication where A is distributed in a (*, BLOCK) fashion.

HPF version of sparse storage CG (CSR format):

```
      REAL, dimension(1:nz) :: A
      INTEGER, dimension(1:nz) :: col
      INTEGER, dimension(1:n+1)  :: row
      REAL, dimension(1:n) :: x, r, p, q
!HPF$ PROCESSORS :: PROCS(NP)
!HPF$ DISTRIBUTE (BLOCK) :: q, p, r, x
!HPF$ DISTRIBUTE A(BLOCK)
!HPF$ DISTRIBUTE col(BLOCK)
!HPF$ DISTRIBUTE row(CYCLIC((n+1)/np)

      (usual initialisation of variables)
```

HPF version of sparse storage CG (CSR format) - Ctd.

```
DO k=1,Niter
  rho0 = rho
  rho = DOT_PRODUCT(r, r)  ! sdot
  beta = rho / rho0
  p = beta * p + r     ! saypx

  q = 0.0              ! sparse mat-vect multiply
  FORALL( j=1:n )
    DO i = row(j), row(j+1)-1
      q(j) = q(j) + A(i) * p(col(i))
    END DO
  END FORALL

  alpha = rho / DOT_PRODUCT(p, q)
  x = x + alpha * p    ! saxpy
  r = r - alpha * q    ! saxpy
  IF ( stop_criterion ) EXIT
END DO
```

CG (Sparse Methods) Conclusions:

- key issue for a parallel CG algorithm is matrix-vector routine.
- this must be able to exploit data storage scheme - typically compressed storage of sparse array.
- Current HPF distribution directives only allow arrays to be distributed according to **regular** structures such as BLOCK and CYCLIC.
- Whilst this is adequate for dense or regularly structured problems it does not provide the necessary flexibility for the efficient storage and manipulation of arbitrarily sparse matrices.
- Syntactic additions for HPF-2 may address this deficiency.

NPAC Projects Exploring HPF Applicability:

- Blackhole Binaries
- Weather/Climate Optimal Data Interpolation / Assimilation
- Parallel Shear Flow Code and Acoustic Equations implementation in HPF
- Work with ICASE on TLNS3D code

Overall Conclusions:

- HPF (potentially) allows: faster computation on parallel and distributed computers;
- additional code portability and ease of maintainance by comparison with message-passing implementations.
- Disadvantages (in common with any parallel implementation) over serial implementations are additional temporary data-storage requirements of parallel algorithms.
- HPF compilation systems now actually available (Digital, Portland, APR,...)
- HPF-2 language definition may address many of deficiencies now being uncovered by applications.

Contents of the HPFA Package:

- List of Available Compilers, translators,...
- List of Industrial and Academic application areas with an indication of appropriate software including suitability of High Performance Fortran.
- List of generic exemplar applications codes with discussion of issues of relevance to HPF and HPF+.
- List of papers and books on HPF, Fortran90 and associated parallel computing issues;
- On-line HPF Tutorial;
- Answers to commonly asked questions on HPF;
- Talks and Lectures on HPF.

Online Internet Resources:

- **http://www.npac.syr.edu/hpfa** HPFA Project material at NPAC.
- **http://www.netlib.org/nse/home.html** The National HPCC Software Exchange.
- email: hpfaman@npac.syr.edu