Outline

1. Introduction to Data-Parallelism
2. Fortran 90 Features
3. HPF Parallel Features
4. HPF Data Mapping Features
5. Parallel Programming in HPF
6. HPF Version 2.0

Fortran 90

- **Fortran 90 is a major extension of FORTRAN 77 including:**
  - *All* of FORTRAN 77
  - Syntax Improvements
  - Memory Allocation
  - Array Features
  - New Intrinsics
  - Procedure Interfaces
  - Pointers
  - User-defined Types
- **Adopted by ANSI and ISO in 1991 (hence “F90”)**
  - Fortran 95 expected to be adopted soon
  - Work on Fortran 2000 progressing nicely

Fortran 90 Memory Allocation

- **`ALLOCATABLE` arrays use heap storage**
  - Declared without defined bounds
  - `ALLOCATE` & `DEALLOCATE` statements create and free storage
- **Local arrays use stack storage**
  - Bounds may be non-constant, evaluated at procedure entry
  - E.g., local arrays the same size as a dummy parameter
- **Assumed-shape arrays can pass array sizes**
  - Dummy parameters without declared bounds
  - Compiler passes bounds in a data descriptor
  - Bounds can be queried with intrinsics

Typical Uses of Memory Allocation

Allocating arrays in main program

```
REAL, ALLOCATABLE :: u(:,:), f(:,:), r(:,:)
ALLOCATE( u(0:nx,0:ny), f(0:nx,0:ny), &
   r(0:nx,0:ny) )
```

Passing assumed-shape arrays

```
INTERFACE
   SUBROUTINE residual(r,u,f)
      REAL r(:,:), u(:,:), f(:,:)
   END SUBROUTINE
END INTERFACE
CALL residual( r, u, f )
CALL residual( r(0:nx:2,0:ny:2), &
   u(0:nx:2,0:ny:2), f(0:nx:2,0:ny:2) )
```

Fortran 90 Array Operations

- **Whole arrays and array subsections can be used in many operations formerly limited to scalar values.**
  - Array arithmetic and assignment
  - `WHERE` (conditional assignment)
  - Vector subscripts
  - Subroutine arguments and return values
- **Subsections specified by triplet notation**
  - `A( lb : ub : step )`
  - Defaults for missing `lb`, `ub`, or `step`
- **Array constructors to produce array-valued constants (with implied `DO`)**

Typical Uses of Array Operations

Scaling

```
REAL u(0:nx,0:ny), fact, avg
u = fact * (u - avg)
```

Computing a residual (version 1)

```
REAL r(0:nx,0:ny), u(0:nx,0:ny), f(0:nx,0:ny)
r(0:nx:jmp,0:ny:jmp)=-u(0:nx:jmp,0:ny:jmp)+ &
   0.25*(CSHIFT(u(0:nx:jmp,0:ny:jmp),1,1) + &
      CSHIFT(u(0:nx:jmp,0:ny:jmp),-1,1) + &
      CSHIFT(u(0:nx:jmp,0:ny:jmp),1,2) + &
      CSHIFT(u(0:nx:jmp,0:ny:jmp),-1,2)) + &
   f(0:nx:jmp,0:ny:jmp)
```

Computing a residual (version 2)

```
REAL r(:,:), u(:,:), f(:,:)
r = - u+0.25*(CSHIFT(u,1,1)+CSHIFT(u,-1,1)+ &
   CSHIFT(u,1,2)+CSHIFT(u,-1,2)) + f
```

Fortran 90 Intrinsics

- **Elemental intrinsics**
  - Most familiar intrinsics can be applied to arrays
  - E.g., `SQRT(A(1:100))` computes 100 square roots
- **Transformational intrinsics**
  - Reductions combine elements in an array (or along one dimension at a time)
  - E.g., `SUM(A)` adds up the elements in `A`
  - Data movement operations perform structured transformations
  - E.g. `TRANSPOSE(A)` transposes `A` (2-D only)
- **Array query intrinsics**
  - Useful for assumed-shape arrays
  - E.g., `SIZE(A,1)` gives the extent of `A` in the first dimension

Typical Uses of Intrinsics

Initialization

```
REAL, ALLOCATABLE :: u(:,:)
CALL RANDOM_SEED
CALL RANDOM_NUMBER(u)
```

Computing the error

```
REAL r(0:nx,0:ny)
error_value = SQRT( SUM(r*r) )
```

Prolongation

```
REAL u(0:nx,0:ny)
u(0:nx:jmp2,jmp:ny:jmp2) = &
   u(0:nx:jmp2,jmp:ny:jmp2)+&
   0.5*CSHIFT(u(0:nx:jmp2,jmp2:ny:jmp2),1,2)
```

Fortran 90 Procedure Interfaces

- **Explicit and implicit interfaces**
  - Explicit: in the same compilation unit, or imported interface
    - Required for certain uses such as `POINTER` dummies and assumed-shape array dummies
  - Implicit: as in FORTRAN 77
- **INTERFACE blocks provide explicit interfaces**
  - Define types of parameters and return values
- **MODULE provides type-safe libraries**
  - Contains declarations, functions, and type definitions
  - All interfaces collected in one place
  - `PRIVATE` data and procedures allowed
  - USE imports the interfaces

Typical Uses of Procedure Interfaces

Option 1: Use Fortran 77-style arguments

```
CALL residual(r, u, f, nx, ny)
! No way to pass every other element of r
```

Option 2: Use assumed-shape arrays

```
INTERFACE
   SUBROUTINE residual(r,u,f)
      REAL r(:,:), u(:,:), f(:,:)
   END SUBROUTINE
END INTERFACE
CALL residual( r, u, f )
CALL residual( r(0:nx:2,0:ny:2), &
   & u(0:nx:2,0:ny:2), f(0:nx:2,0:ny:2) )
```

Hints for Using Fortran 90

- **Declare arrays to be the natural size and shape**
  - `ALLOCATE` in main routine (or when size is set)
  - Automatic local arrays
  - Assumed shape dummies (controversial)
- **Array operations are great for data parallelism**
  - But they don't cover everything…
- **Use `MODULE` and `INTERFACE`**
  - Consistency checking for you
  - Extra information for the compiler
- **Complicated data structures are possible**
  - Unclear what compilers will optimize well