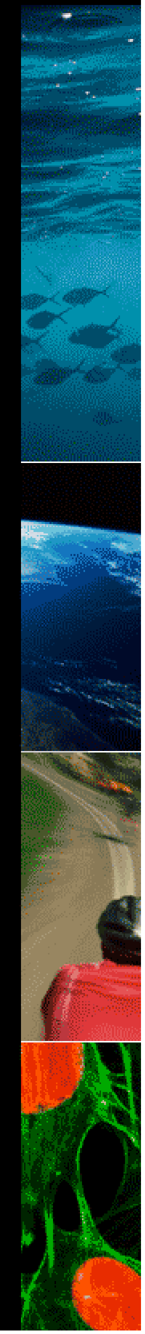
CRAY The Supercomputer Company

Programming for High Performance Computing in Modern Fortran

Bill Long, Cray Inc.

17-May-2005



Concepts in HPC

- Efficient and structured layout of local data
  - modules and allocatable arrays
- Efficient operations on local data
  - array operations and loop nest optimizations
- Efficient and easy to use parallel programming
  - co-arrays
- Tips on things to avoid

Modules - replacement for common and include

- Modules have specification and procedure parts
- Specification part may contain
  - named constants (parameters)
  - type definitions
  - data declarations (global data, like common blocks)
    - allocatable objects
    - private objects (names are public or private)
    - public objects
    - protected data   (data values are protected)
  - explicit procedure interfaces and generic interfaces

Modules continued

- Procedure parts contain module procedure definitions
  - Interfaces for module procedures are explicit
  - Module procedures have access to all objects in the specification part of the module by host association
  - Protected data definable only by a local module procedure
  - Inlining of module procedures is default for X1 compiler

- Public objects accessed by a USE statement

Allocatable arrays

- Allocatable arrays provide dynamic memory management
- More efficient than pointers because aliasing issues avoided
- Makes better use of memory; avoids oversized arrays

(Arrays in general are a good thing for HPC, of course)

Old Fortran example

```
integer,parameter :: n1 = 207, n2 = 331, n3 = 501
real(8) :: grid1(n1,n2,n3),grid2(n1,n2,n3)
common /gridcb/ grid1,grid2, m1,m2,m3

call read_data(filename)

do k=1,m3
    do j=1,m2
        do i=1,m1
            grid2(i,j,k) = grid1(i,j,k)
        end do
    end do
end do
```

Example using modules and allocatable arrays

```
module grids
    real(8),allocatable,dimension(:,:,:)  :: grid1,grid2
    integer,protected :: m1,m2,m3
contains
    subroutine read_data(filename)
       ! reads in m1,m2,m3 from file
       allocate(grid1(m1,m2,m3),grid2(m1,m2,m3))
       ! read in the data for grid1
    end subroutine read_data
end module grids
!----------------------
use grids
  call read_data(filename)
  grid2(:,:,:) = grid1(:,:,:)
```

Advantages of the new style

- Use only the amount of memory needed
- Assignment performance is much better - loop collapse
- Values of m1,m2,m3 are protected against accidental definition
- Only write the declarations once, then USE in each program unit needing access to the data
- read routine and the data are packaged together for easier maintenance

Disadvantage of the new style

- If you modify the code in the read_data subroutine, the module changes. The make file will cause all files that USE the module to be recompiled.

Submodules (f08 feature)

- Parent module contains procedure interface information
- Actual code for the procedure in a submodule of the parent
- Use separate files for parent and various submodules
- Programmer only USE's the parent
- Changes to procedure code avoids compile cascade
- Simplifies management of very large projects with many programmers

Parallel programming - Co-arrays (f08 feature)

- With the addition of co-arrays Fortran becomes a fundamentally parallel language.

Parallel programming models

- Shared memory models
  - OpenMP
  - autotasking
  - multithreading
- Distributed memory models
  - MPI and PVM
    - general, hard to use, performance can be poor
  - shmem
    - single sided, take advantage of symmetric addresses
  - co-arrays
    - syntax implementation of shmem

co-array syntax - basic

Program consists of multiple identical IMAGES (SPMD model)

real :: X(100)[*], S[*]

X is the array on this image
X(:)[4] is the array on image # 4

S is the scalar on this image
S[4] is the scalar on image #4

THIS_IMAGE() returns the number of this image
NUM_IMAGES() returns the number of images

co-array syntax - allocatable co-arrays

Allocatable co-arrays are allowed. Must allocate on each image with the same size.

```
real :: A(:)[:]

allocate(A(100)[*])
```

The allocate contains an implicit barrier. Unsaved allocatable co-arrays are deallocated on exit from a procedure; the implicit deallocate also contains a barrier.

co-array syntax - pointer components

Pointer components provide access to non-symmetric data. Useful for accessing remote dummy arguments

```
type ca_pointers
    real,pointer :: a(:),b(:,:)
end type ca_pointers

type(ca_pointers) :: image[*]

image%a => a   !set up local pointers to local data
image%b => b
call sync_all()
c(:) = image[4]%a(:)  ! get values in A on image 4
```

co-array syntax - allocatable components

Allocatable components provide a way to share objects with different sizes on each image

```
type ca_vla
   real,allocatable :: V(:)
end type ca_vla

type(ca_vla) :: image[*]

allocate(image%V(n))  ! local allocate - no barrier
call sync_all()
v1 = image[4]%V(1)
```

sync routines

With a few exceptions, the images execute asynchronously.
If syncs are needed, the user supplies then explicitly.

```
call sync_all()   ! barrier on all images

call sync_team(team_list) ! barrier on the images listed in the
                          ! team_list array.

call flush_memory()  ! forces memory operation ordering on local image.
                     ! included in sync_all and sync_team

call notify_team(team_list) ! check into a barrier, but do not wait
call wait_team(team_list)   ! wait for others to check into a barrier
```

Advantages of co-arrays

Very easy to write code - the communication is explicit in the syntax.

No initialize or finalize routines are required

Very few function names to remember - mostly sync routines

Makes use of the Fortran language rules

- built in support for derived types and all intrinsic types
- supports strided and gather/scatter data transfers simply
- type conversions on assignment follow Fortran rules

Optimized implementations can reduce communication overhead

Example code - MPI version

```
if(Left>=0) then
  call MPI_IRecv(neg_f,(my*mz*iorder/2),MPI_REAL8,Left,1,gcomm,&
                 req(1),ierr)
  call MPI_ISend(f(1,1,1),1,xrows_type,Left,2,gcomm,req(2),ierr)
endif

if(Right>=0) then
  call MPI_IRecv(pos_f,(my*mz*iorder/2),MPI_REAL8,Right,2,gcomm,&
                 req(3),ierr)
  nm = mx + 1 - iorder/2
  call MPI_ISend(f(nm,1,1),1,xrows_type,Right,1,gcomm,req(4),ierr)
endif
```

Example code - Co-array version

```
call sync_all()
if(Left>0) then
   neg_f(:,:,:)[Left] = f(1:iorder/2,:,:)
endif

if(Right>0) then
   nm = mx + 1 - iorder/2
   pos_f(:,:,:)[Right] = f(nm:nm+iorder/2-1,:,:)
endif
call sync_all()
```

Happy Users

“Very cool!   As far as I am concerned, co-array programming is easy even when retro-fitting it to another code. It makes sense too. “

- ORNL

“It is such an intuitively obvious extension of Fortran90 for parallel programming that I think everybody should be using it.”

-ARSC

Implementation considerations

SMP machines:

Treat co-dimensions like extra ordinary dimensions

Distributed memory machines with global addressing:

Modify the addresses of the remote data with image number and just issue load and store instructions

Clusters:

Compiler converts syntax into shmem calls (worst case).
Still have all the ease of use advantages.

Implementation on single cpu systems

- THIS_IMAGE() == 1
- NUM_IMAGES() == 1
- Ignore the [ ]
- Ignore the sync routines
- Only need to parse the syntax

Additional co-arrays reading

- AHPCRC Bulletin 2004 - Vol. 14 No. 4 and references therein, especially the article by Jef Dawson (Their web site, www.ahpcrc.org/publications, does not have this one up yet.)
- Cray's Fortran Language Reference manual for X1.
- J3 paper 05-183r1 from meeting 172 at j3-fortran.org

Misc programming tips for performance

- \begin{soap_box}
- Avoid pointers if target is fixed - use allocatable instead
- Do not manually unroll loops - let the compiler do it for you
- Avoid BLAS1 calls - these are one-line array assignments
- Avoid really long argument lists
- Just say no to MPI
- \end{soap_box}

Really Last Slide!

- Thanks to Ted Stern for valuable discussions
- Thanks to ORNL users for the code examples (abstracted here)
- Contact information:

Bill Long
longb@cray.com