Coscalars

A suggestion for supporting distributed structures in Fortran
Declaration / establishment

- **coarray:**
  - exists on every image
  - dynamic memory: requires program-wide synchronization

- **coscalar:**
  - exists on exactly one image ("host image")
  - dynamic memory: exactly one image allocates
    - becomes the host image
  - integer :: n[
  - real, allocatable :: x[], y(:)[
    - allocate(x[5], (y(20))[7])
    - no sync, atomic semantics
      - (invoke ALLOCATED())
    - same image deallocates

---

x - scalar coscalar
y - array coscalar

Corank is zero
Definition / Reference

• Explicit bracket
  • always present („statistics“)
  • all accesses coindexed

• Synchronization rule
  • same as for coarrays

```plaintext
integer :: n[

if (this_image(n) == this_image()) then
  n[] = ...
  sync images(*)
else
  sync images(this_image(n))
  ... = n[] ...
end if
```
Coscalar pointer

- add the attribute
  
  ```
  real, pointer :: p[
  
  • itself a coscalar
  • target: must be coindexed
  ```

- expensive operations → 3 (or more) images involved
  
  ```
  real :: t1[*]
  real :: t2[
  ```

- require TARGET attribute?
  
  ```
  if (X) p[] => t1[5]
  sync images (X)
  ...
  = p[]
  sync all
  if (...) p[] => t2[
  ```

- host image of pointer
- image X executing the pointer assignment
- host image of target
  (to some extent, have this problem also for coarrays)
Distributed binary tree

• all entities must be coscalars (then %lock is)
• support a tasking-like, recursive programming style
  • lock needed for population, not read-only traversal
  • therefore preference for read operations
• allocation and deallocation remain purely local operations
• with coarrays, implementation much more clumsy
Rice „CAF 2.0“ Copointers

A more full-featured approach
Declaration / establishment

- uses the COPOINTER attribute
  
  ```
  real, dimension(:), &
  copointer :: px
  ```

  - entity exists on one image only

- target must have the COTARGET attribute
  
  ```
  real, dimension(10), &
  cotarget :: x
  ```

  - exists on one image only

- Copointer association
  
  ```
  if (this_image() == 1) &
  px => x
  ```

- additional attributes like CONTIGUOUS are also allowed

- dynamic (de)allocation of anonymous cotarget (shared area, one image only) is possible
Copointers to coarrays

- local portion
  - \( \text{real, dimension}(::), \text{copointer} :: px \)
  - \( \text{real, dimension}(10), \text{cotarget} :: x[*] \)
  - \( px \rightarrow x \)

- coindexed entity
  - \( px \rightarrow x[9] \)

  (remote operation wrt. copointer location)

Copointers to local pointers

- \( \text{real, dimension}(::), \text{pointer} :: lpx \)
- \( lpx \rightarrow x \quad \text{!! cotarget implies target (?)} \)
- \( ... \)
- \( px \rightarrow lpx \quad \text{!! converts local pointer to copointer} \)
Local casts

- remote RHS for local LHS remains disallowed
  \[
  lpx \rightarrow px[] \quad ! \text{forbidden}
  \]

- Only allowed if RHS target is local
  
  ```
  if (imageof(px) == this_image()) then
      lpx \rightarrow px \quad ! \text{OK}
  end if
  ```

  this also illustrates how target location is identified
Copointers that are coarrays

- gives you a bunch of num_images() copointers, each hosted on an image of its own

\[
\text{real, dimension(:), copointer :: pxc[*]}
\]

- make each of these point to a coarray

\[
\text{real, dimension(10), cotarget :: xc[*]}
\]

\[
\text{pxc => xc ! all images execute}
\]
Referencing / defining

- Requires the co-dereference operator

```fortran
real, dimension(:), copointer :: px
real, dimension(10) :: a

if (this_image() == 1) then
    allocate(px(4))
    px(1)[] = a(5)
    px(2:)[] = a(1:3)
end if

! same for RHS
```

- Exception: target is local to the executing image
Conclusion

- Feature is for functionality, not for performance
- better support for implementation of object-based parallel software patterns
  - mostly of the kind write rarely, read often
  - e.g. load balancing algorithms
- Locks, Events may be more flexibly used