Advanced Features

PRACE Autumn School, October 29th 2010

Parallel Programming with Coarray Fortran

CRAY THE SUPERCOMPUTER COMPANY | epcc

Advanced Features: Overview

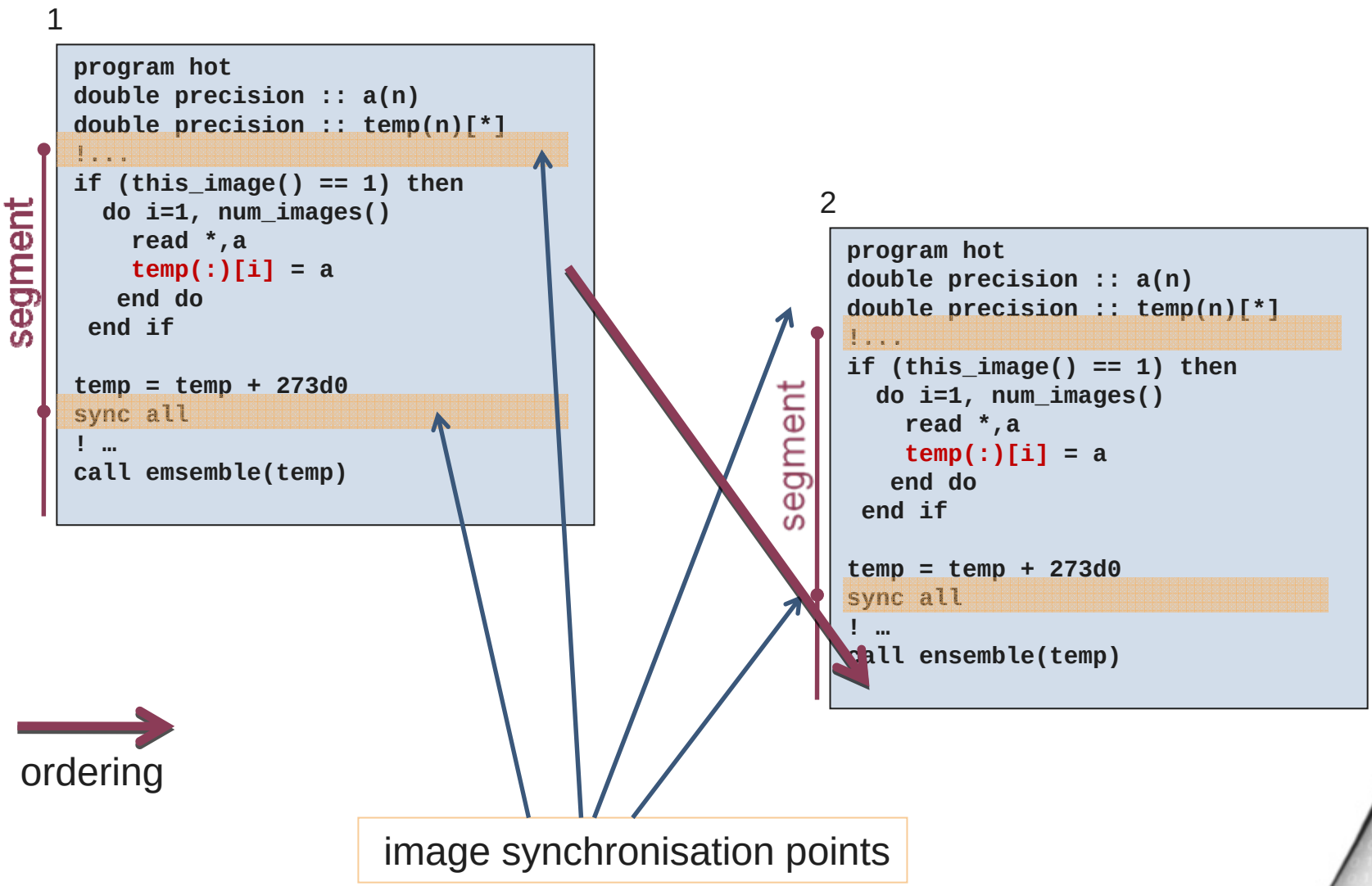
- Execution segments and Synchronisation
- Non-global Synchronisation
- Critical Sections
- Visibility of changes to memory
- Other Intrinsics
- Miscellaneous features
- Future developments

More on Synchronisation

- We have to be careful with one-sided updates
  - If we read remote data, was it valid?
  - Could another process send us data and overwrite something we have not yet used?
  - How do we know when remote data has arrived?
- The standard introduces **execution segments** to deal with this: segments are bounded by image control statements

- If a variable is defined in a segment, it must not be referenced, defined, or become undefined in another segment unless the segments are ordered

Execution Segments



Synchronisation mistakes

- This code is wrong

```
subroutine allreduce_max_getput(v,vmax)
  double precision, intent(in)  :: v[*]
  double precision, intent(out) :: vmax[*]
  integer i
  sync all

  vmax=v
  if (this_image()==1) then
    do i=2,num_images()
      vmax=max(vmax,v[i])
    end do
    do i=2,num_images()
      vmax[i]=vmax
    end do
  end if
  sync all
```

Synchronisation mistakes

- It breaks the rules

```
subroutine allreduce_max_getput(v,vmax)
  double precision, intent(in)  :: v[*]
  double precision, intent(out) :: vmax[*]
  integer i
  sync all

  vmax=v
  if (this_image()==1) then
    do i=2,num_images()
      vmax=max(vmax,v[i])
    end do
    do i=2,num_images()
      vmax[i]=vmax
    end do
  end if
  sync all
```

Synchronisation mistakes

- This is ok

```
subroutine allreduce_max_getput(v,vmax)
  double precision, intent(in)  :: v[*]
  double precision, intent(out) :: vmax[*]
  integer i
  sync all

  if (this_image()==1) then
    vmax=v
    do i=2,num_images()
      vmax=max(vmax,v[i])
    end do
    do i=2,num_images()
      vmax[i]=vmax
    end do
  end if
  sync all
```

More about `sync all`

- Usually all images execute the same sync all statement

- But this is not a requirement..
  - Images execute different code with different **`sync all`** statements
  - All images execute the first sync all they come across and….
    - this may match an arbitray sync all on another image
    - causing incorrect execution and/or deadlock

- Need to be careful with this ‘feature’
  - Possible to write code which doesn’t deadlock but gives wrong answers

More about **sync all**

- e.g. Image practical: wrong answer

```
! Do halo swap, taking care at the upper and lower picture boundaries

    if (myimage < numimage) then
      oldpic(1:nxlocal, nylocal+1) = oldpic(1:nxlocal, 1)[myimage+1]

    sync all
    end if

! ... and the same for down halo
! Now update the local values of newpic
  ...

! Need to synchronise to ensure that all images have finished reading the
! oldpic halo values on this image before overwriting it with newpic

  sync all
  oldpic(1:nxlocal,1:nylocal) = newpic(1:nxlocal,1:nylocal)

! Need to synchronise to ensure that all images have finished updating
! their oldpic arrays before this image reads any halo data from them

  sync all
```

All images NOT executing this sync all

All images ARE executing this sync all

More about **`sync all`**

- **`sync images(`*`imageList`*`)`**
  - Performs a synchronisation of the image executing **`sync images`** with each of the images specified in ***`imageList`***
  - ***`imageList`*** can be an array or a scalar

```
! Do halo swap, taking care at the upper and lower picture boundaries

    if (myimage < numimage) then

      oldpic(1:nxlocal, nylocal+1) = oldpic(1:nxlocal, 1)[myimage+1]
      sync images(myimage+1)

    end if

    if (myimage > 1) then

      oldpic(1:nxlocal, 0) = oldpic(1:nxlocal, nylocal)[myimage-1]
      sync images(myimage-1)

    end if
```

CRAY THE SUPERCOMPUTER COMPANY | epcc

Other Synchronisation

- Critical sections
  - Limit execution of a piece of code to one image at a time
  - e.g. calculating global sum on master image

```
integer :: a(100)[*]
integer :: globalSum[*] = 0, localSum
... ! Initialise a on each image

localSum = SUM(a) !Find localSum of a on each image

critical
   globalSum[1] = globalSum[1] + localSum
end critical
```

Other Synchronisation

- **`sync memory`**
  - Coarray data held in caches/registers made visible to all images
    - requires some other synchronisation to be useful
    - unlikely to be used in most coarray codes

- **`sync memory`** implied for **`sync all`** and **`sync images`**

Other Synchronisation

- **`lock`** and **`unlock`** statements
  - Control access to data defined or referenced by more than one image
    - as opposed to **`critical`** which controls access to lines of code
    - **`USE iso_fortran_env`** module and define coarray of **`type(lock_type)`**
  - e.g. to lock data on image 2

```
type(lock_type) :: qLock[*]

lock(qLock[2])
!access data on image 2
unlock(qLock[2])
```

Other Intrinsic functions

- **`lcobound(z)`**
  - Returns lower cobounds of the coarray **`z`**
  - **`lcobound(z,dim)`** returns lower cobounds for codimension **`dim`** of **`z`**
- **`ucobound(z)`**
  - Returns upper cobound of the coarray **`z`**
  - **`lcobound(z,dim)`** returns upper cobound for codimension **`dim of z`**

- **`real :: array(10)[4,0:*]`** on 16 images
  - **`lcobound(array)`** returns **`[ 1, 0 ]`**
  - **`ucobound(array)`** returns **`[ 4, 3 ]`**

More on Cosubscripts

- `integer :: a[*]` on 8 images
  - cosubscript `a[9]` is not valid

- `real :: b(10)[3,*]` on 8 images
  - `ucobounds(b)` returns `[ 3, 3 ]`
  - cosubscript `b[2,3]` is valid (corresponds to image 8)…
  - …but cosubscript `b[3,3]` is invalid (image 9)

- Programmer needs to make sure that cosubscripts are valid
  - `this_image` returns 0 for invalid cosubscripts

Assumed Size Coarrays

- Codimensions can be remapped to corank greater than 1
  - useful for determining optimal extents at runtime

```
program 2d
real, codimension[*] :: picture(100,100)
integer :: numimage, numimagex, numimagey
numimage = num_images()

call get_best_2d_decomposition(numimage,&
       numimagex, numimagey)
! Assume this ensures numimage=numimagex*numimagey

call dothework(picture, numimagex, numimagey)
...
contains
   subroutine dothework(array, m, n)
   real, codimension[m,*] :: array(100,100)
   ...
end subroutine dothework
```

I/O

- Each image has its own set of input/output units
- units are independent on each image
- Default input unit is preconnected on image 1 only
  - **`read *,…`** , **`read(*,…)…`**
- Default output unit is available on all images
  - **`print *,…`** , **`write(*,…)…`**
  - It is expected that the implementation will merge records from each image into one stream

Program Termination

- STOP or END PROGRAM statements initiate *normal termination* which includes a synchronisation step
- An image's data is still available after it has initiated normal termination
- Other images can test for this using STAT= specifier to synchronisation calls or allocate/deallocate
  - test for STAT_STOPPED_IMAGE (defined in ISO_FORTRAN_ENV module)
- The ERROR STOP statement initiates error termination and it is expected all images will be terminated.

Coarray TR

- New coarray features may be described in a Technical Report (TR)
- Work in progress but the areas of discussion are:
  - **image teams**
  - **collective intrinsics for coarrays**
  - file operations by more than one image
  - new atomics
  - coarray pointers and non-symmetric allocation
  - coscalars

TR: TEAMs of Images

- To define a set of images as a TEAM

```
call form_team(team,[ (i,i=1,n,2) ])
```

- To synchronise the team

```
sync team(team)
```

- To determine images that constitute a team

```
images=team_images(team)
```

CRAY THE SUPERCOMPUTER COMPANY | epcc |

TR: Collective intrinsic subroutines

- Collectives, with in/out arguments, invoked by same statement on all images (or team of images)
- Routines
  - CO_SUM and other reduction operations
  - CO_MINVAL, CO_MAXVAL
  - Possibly more general reduction
- Arguments include SOURCE, RESULT, TEAM
- Still discussion on need for implicit synchronisation and argument types (for example non-coarray arguments)


CRAY
THE SUPERCOMPUTER COMPANY
epcc