Co-Array Fortran

Phil Russell
Alex Liber
Micah Wendell

What It Is

- Formally called F--,
- Small set of semantic extensions to Fortran 95
- Simple syntactic extension to Fortran 95
- Single Program Multiple Data, SPMD, parallel processing

What It Is

- Robust, efficient parallel language.
- Requires learning only a few new rules.
- Rules handle two fundamental issues:

    Work distribution

    Data distribution.

Work Distribution

- A single program is replicated a fixed number of times
- Each replication has its own set of data objects.
- Each replication of the program is called an image.
- Each image executes asynchronously

Work Distribution

- The normal rules of Fortran apply
- The execution path may differ from image to image.
- The programmer determines the actual path for the image with
  - A unique image index
  - Normal Fortran control constructs
  - Explicit synchronizations.

Work Distribution

- Code between synchronizations

  The compiler is free to use all its normal optimization techniques, as if only one image is present

Data Distribution

- Specify the data relationships
- One new object, the co-array, is added to the language
- An example…

Data Distribution

```
REAL, DIMENSION(N)[*] :: X,Y
    X(:) = Y(:)[Q]
```

The above statement declares that each image has two real arrays of size N. If Q has the same value on each image, the effect of the assignment statement is that each image copies the array Y from image Q and makes a local copy in array X.

Data Distribution

- (index) follow the normal Fortran rules within one memory image.
- [index] provide access to objects across images and follow similar rules.
- [bounds] in co-array declarations follow the rules of assumed-size arrays since co-arrays are always spread over all the images.

Data Distribution

- The programmer uses co-array syntax only where it is needed
- A co-array reference with no square brackets is a reference to the object in the local memory
- Co-array syntax should appear only in isolated parts of the code
- If not, too much communication among images?
  - Flags compiler to avoid latency
  - Flags programmer to rethink

Extended Fortran 90 Array Syntax

- A way of expressing remote memory operations. Here are some simple examples:

- X = Y[PE] get from Y[PE]
- Y[PE] = X put into Y[PE]
- Y[:] = X broadcast X
- Y[LIST] = X broadcast X over subset of PE's in array LIST
- Z(:) = Y[:] collect all Y
- S = MINVAL(Y[:]) min (reduce) all Y
- B(1:M)[1:N] = S S scalar, promoted to array of shape (1:M,1:N)

Input/Output

- Input/output problem with SPMD programming models
- Fortran I/O assumes dedicated single-process access to an open file

  Often violated when it is assumed that I/O from each image is completely independent.

Input/Output

- Co-Array Fortran includes only minor extensions to Fortran 95 I/O,
- All the inconsistencies of earlier programming models have been avoided
- There is explicit support for parallel I/O.
- I/O is compatible with both process-based and thread-based implementations.

Other Fortran 95 additions: Several Intrinsics

- NUM_IMAGES() returns the number of images,
- THIS_IMAGE() returns this image's index between 1 and NUM_IMAGES()
- SYNC_ALL() is a global barrier
- To only wait for the relevant images to arrive. SYNC_ALL(WAIT=LIST)

More Intrinsics

- SYNC_TEAM(TEAM=TEAM)
- SYNC_TEAM(TEAM=TEAM,WAIT=LIST)
- START_CRITICAL and END_CRITICAL

Adding Synch Functionality

- SYNC_MEMORY().

  This routine forces the local image to both complete any outstanding co-array writes into ``global'' memory and refresh from global memory any local copies of co-array data it might be holding (in registers for example).

  Image synchronization implies co-array synchronization.

- A call to SYNC_MEMORY() is rarely required

  Implicitly called before and after virtually all procedure calls including Co-Array's built in image synchronization intrinsics.

Image and co-array synchronization

- Example: exchanging an array with your north and south neighbors:

```
    COMMON/XCTILB4/ B(N,4)[*]
    SAVE  /XCTILB4/

    CALL SYNC_ALL( WAIT=(/IMG_S,IMG_N/) )
    B(:,3) = B(:,1)[IMG_S]
    B(:,4) = B(:,2)[IMG_N]
    CALL SYNC_ALL( WAIT=(/IMG_S,IMG_N/) )
```

Array Exchange Synchronization Explained

- The first SYNC_ALL waits until the remote B(:,1:2) is ready to be copied
- The second waits until it is safe to overwrite the local B(:,1:2).
- Only nearest neighbors are involved in the sync.
- It is always safe to replace SYNC_ALL(WAIT=LIST) calls with global SYNC_ALL() calls

  Often is significantly slower.

  Either the preceding or succeeding synchronization may be avoidable.

Synch Optimization

- The majority of remote co-array access optimization is minimizing the synchronization
  - Frequency of synchronization
  - Cover the minimum number of images
- On machines without global memory hardware, array syntax (rather than DO loops) should always be used for remote memory operations
- Copying co-array's into local temporary buffers before they are required might be appropriate

Data Parallel Cumulative Sum

- In data parallel programs, each image is either performing the same operation or is idle.
- For example here is a data parallel fixed order cumulative sum:

```
REAL SUM[*]
CALL SYNC_ALL( WAIT=1 )

DO IMG= 2,NUM_IMAGES()
    IF (IMG==THIS_IMAGE()) THEN
        SUM = SUM + SUM[IMG-1]
    ENDIF
    CALL SYNC_ALL( WAIT=IMG )
ENDDO
```

Data Parallel Performance Critique

- SYNC_ALL waiting on just the active image improves performance
- still NUM_IMAGES() global sync

An Alternative to Data Parallel

- A better alternative may be to minimize synchronization by avoiding the data parallel overhead entirely:

```
REAL SUM[*]
ME = THIS_IMAGE()
IF (ME.GT.1) THEN
        CALL SYNC_TEAM( TEAM=(/ME-1,ME/) )
        SUM = SUM + SUM[ME-1]
ENDIF
IF (ME.LT.NUM_IMAGES()) THEN
        CALL SYNC_TEAM( TEAM=(/ME,ME+1/) )
ENDIF
```

Alternative Performance Analysis

- Now each image is involved in at most two sync's: the images just before and just after it in image order.
- The first SYNC_TEAM call on one image is matched by the second SYNC_TEAM call on the previous image.

Benefits (or: In Summary)

- The Co-Array Fortran synchronization intrinsics can :
  - Improve the performance of data parallel algorithms
  - Provide implicit program execution control as an alternative to the data parallel approach.

Amusement

```
depfile='.deps/DeclarationAnalysisPass.Po' tmpdepfile='.deps/DeclarationAnalysis
Pass.TPo' \
depmode=gcc3 /bin/sh ../../depcomp \
g++ -DPACKAGE_NAME=\"Rice\ Co-array\ Fortran\ Compiler\" -DPACKAGE_TARNAME=\"ric
e-co-array-fortran-compiler\" -DPACKAGE_VERSION=\"0.9\" -DPACKAGE_STRING=\"Rice\
 Co-array\ Fortran\ Compiler\ 0.9\" -DPACKAGE_BUGREPORT=\"\" -DPACKAGE=\"rice-co
-array-fortran-compiler\" -DVERSION=\"0.9\" -DSTDC_HEADERS=1 -DHAVE_SYS_TYPES_H=
1 -DHAVE_SYS_STAT_H=1 -DHAVE_STDLIB_H=1 -DHAVE_STRING_H=1 -DHAVE_MEMORY_H=1 -DHA
VE_STRINGS_H=1 -DHAVE_INTTYPES_H=1 -DHAVE_STDINT_H=1 -DHAVE_UNISTD_H=1 -DHAVE_LI
MITS_H=1 -DHAVE_STDLIB_H=1 -DHAVE_STRING_H=1 -DHAVE_STRINGS_H=1 -DHAVE_UNISTD_H=
1 -DHAVE__BOOL=1 -DHAVE_STDBOOL_H=1 -DHAVE_BZERO=1 -DHAVE_MEMCHR=1 -DHAVE_STRCAS
ECMP=1  -I. -I. -I../../src -I../../src/cafc -I../../src/cafc/codegen -I../../sr
c/analyzer -I../../src/common -I../../src/doc -I../../src/logical-info -I../../s
rc/placeholders -I../../src/placeholders/ARMCI_PH -I../../src/placeholders/gener
ic -I../../src/utilities -I../../src/utilities/annotation -I../../src/utilities/
ast -I../../src/utilities/iterators -I../../src/utilities/md5 -I/project/cafc/Op
en64/osprey1.0/include -I/project/cafc/Open64/osprey1.0/common/com -I/project/ca
fc/Open64/osprey1.0/common/util -I/project/cafc/Open64/osprey1.0/common/com/ia64
   -D_BSD_SOURCE -LANG:std -g  -c -o DeclarationAnalysisPass.o `test -f '../../s
rc/cafc/DeclarationAnalysisPass.cc' || echo './'`../../src/cafc/DeclarationAnaly
sisPass.cc
../../src/cafc/DeclarationAnalysisPass.cc: In member function `void
   DeclarationAnalysisPass::analyzeCoArrayDeclarations(ST_TAB*,
   PUAnalysisResults*)':
../../src/cafc/DeclarationAnalysisPass.cc:149: `Set_ST_is_deleted' undeclared
   (first use this function)
../../src/cafc/DeclarationAnalysisPass.cc:149: (Each undeclared identifier is
   reported only once for each function it appears in.)
make[2]: *** [DeclarationAnalysisPass.o] Error 1
make[2]: Leaving directory `/project/cafc/cafc/build/cafc'
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory `/project/cafc/cafc/build'
make: *** [all-recursive] Error 1
-bash-2.05b$ _
```