CAF Validation Test Suite Instructional Guide

Siddhartha Jana, Deepak Eachempati

HPCTools, University of Houston,
Texas, 77004
sidjana,dreachem@cs.uh.edu

The purpose of this guide is to help the reader be familiar with the list of configuration parameters(Sec 1), method of validation of the tests(Sec 2), test files(Sec 3), and results of the some of the different CAF compilers [OpenUH(v3.0.x), Cray, Intel(v13.1) and G95(v0.93)] (Sec 4).

1 Locating and configuring the suite

The validation suite comes bundled along with the performance test suite. For this article,

- SUITE_ROOT = the parent directory of the perfomance and validation test suite
- VALIDATION_PATH = $SUITE_ROOT/validation

1.1 Configuration params

Table 1 lists the different options that need to be set for the test-suite. These options can be initialized in the CONFIG file in the "config" directory. The make.def version of this file is auto-generated by the test suite using the script config2makedef.sh (same directory).

2 Categories of Tests

This section outlines the different categories of tests within the suite that can be used to evaluate a CAF compiler implementation. The primary gaol is to determine the extent to which it supports the coaray-features of the Fortran 2008 standard in accordance with ISO/IEC 1539-1:2010 (E)[1].

There are three main categories of tests in the test suite. Each category is located in a different subdirectory within $SUITE_ROOT, and all the tests within the same category can be executed using the makefile.

Feature Tests These tests can be found under $SUITE_ROOT/feature_tests. The tests under this category verify the support and correctness of the implementations of the basic constructs/semantics of CAF. The tests concentrate on the verification of the correctness of contiguous and strided remote read/write operations, the coarray syntax, the use of strided co-subscript notation, coarrays of different data types and image-query intrinsics.

Table 1. List of configuration parameters. Here *test_type*= CONF or FEATURE or FEW or FAULT

Parameter	Description	Compiler specific(y/n)
BIN_PATH	The path to dump all the executables.	NO
NPROCS	Number of images to launch	NO
NITER	Number of times the test is repeated	NO
SLEEP	This is used to intentionally slow down certain images to cause race conditions while testing certain constructs	NO
TIMEOUT	This parameter is passed to the perl script call timedexec.pl which ends processes which exceed the given execution time. This is helpful for backing out while executing tests which deadlock due to incorrect implementation	NO
COMPILER	compiler name	YES
FC	command to invoke the compiler	YES
FFLAGS	Flags passed to the compiler. The necessary flags include the options to enable the macro preprocessor and define the macros - NPROCS, NITER and SLEEP.	YES
LAUNCHER	command to launch multiple images	YES
EXEC_OPTIONS	Flags passed to the launcher after the executable name. Not so common.	YES
FFLAGS_CROSS	Flags passed to the compiler while executing the cross tests. Generally the value include all the options listed for FFLAGS, plus the flag to define the CROSS_ macro in the tests.	YES
*test_type*_COMPILE_PATH	Path to dump the messages generated by the compiler.	NO
*test_type*_EXEC_PATH	Path to dump the output of the executables / the messages generated by the compiler or runtime.	NO
*test_type*_TEST_PATH	Path to dump the output of the test results.	NO
*test_type*_LOG_PATH	Path to dump the output of the test results.	NO

Confidence Tests These tests can be found under $SUITE_ROOT/confidence_tests. All tests in this category verify CAF constructs that aid in maintaining consistency and synchrnization among images. The tests intentional compute intensive loops and forced CPU - idling (using calls to the sleep() intrinsic) for simulating delays in progress of certain images. This enforces an increase in the likelihood of inconsistent states (and races) in case of incorrect implementations of the constructs being tested.

Due to the nondetermistic nature of unsynchronized images, configuration parameters(as defined under $SUITE_ROOT/config/CONFIG*) can be modified to determine the level of confidence with which a test passes a given test case. This test-category uses *cross*-tests to evaluate the confidence of the test-reports. Such *cross*- tests include the exact same code like the original version but with certain statements deleted/replaced to identify the change in behavior of the tests.

Note on code design: Most of the time,the modifications to the original code are in the form of the absence of the CAF statements which are being tested in that test case. In order to reduce redundant code, we chose to include the *cross*-test version and the original version all in one file, using conditional '#ifdef's and macros (e.g. CROSS_) .

Detecting/Reporting of errors

- Every test includes a module called 'crosstest'.
- The module 'crosstest' (defined in file testmofule.f90) includes the declaration of an integer scalar coarray called 'cross_err' which is modified by image with rank 1 on detecting an error. It also contains two subroutines - calc_ori and calc. The former returns the test result when the specific construct is being tested. The latter is called by the cross-test version and returns the confidence with which the original test passed.

Fault Tests These tests can be found under $SUITE_ROOT/fault_tests. These test the correctness of the implementation of specifiers used for handling normal and error termination. These tests include testing the support of statevariables that flag the execution state of other images. Here's an excerpt describing the 2 states of image execution-termination [from Page 23, sec. 13 of "ISO/IEC JTC1/SC22/WG5 N1824 "[2]]:

" ... It seems natural to allow all images to continue executing until they have all executed a stop or end program statement, provided none of them encounters an error condition that may be expected to terminate its execution. This is called normal termination. On the other hand, if such an error condition occurs on one image, the computation is flawed and it is desirable to stop the other images as soon as is practicable. This is called error termination.

Normal termination occurs in three steps: initiation, synchronization, and completion. An image initiates normal termination if it executes a stop or end program statement. All images synchronize execution at the second step so that no image starts the completion step until all images have finished the initiation step. The synchronization step allows its data to remain accessible to the other images until they all reach the synchronization step. Normal termination may also be initiated during execution of a procedure defined by a C companion processor[a.k.a. C compiler]

An image initiates error termination if it executes a statement that would cause the termination of a single-image program but is not a stop or end program statement. This causes all other images that have not already initiated error termination to initiate error termination. Within the performance limits of the processor.s ability to send signals to other images, this propagation of error termination should be immediate. The exact details are intentionally left processor dependent.... "

Specific Tests If the user wants to evaluate a CAF implementation in terms of the support to specific types of tests, the names of the tests can be specified in the file *"test_file"* in the directory -$SUITE_ROOT/few_tests.

3 List of Tests

The tests are listed in Tables 2,3, and 4. The test-files are named using the following convention:

"<construction_type>_<section_number>.f90", where,

- *construction_type* is the semantic-target of the test.
- *section_number* is the section number in the standard to which the check belongs. This allows or a quick look-up of the exact point to which a correct CAF implementation must adhere to.

Table 2. Feature Test files in the UH - CAF Validation Tests suite

File	Description
character_test.f90	CHARACTER coarrays
coarray_2.4.7.6.f90	similar translation of co-subscripts and subscripts
coarray_4.8.R468.f90	reference of coarray without [ ] implies local object
coarray_5.3.6.1.f90	attribute CODIMENSION + remote accesses at single integer/real boundary
dummyargs_12.3.2.2c.f90	explicit shape, assumed size, assumed shape, allocatable dummy args
intrin_13.7.126.f90	NUM_IMAGES() returns the number of images launched
intrin_13.7.165.f90	THIS_IMAGE(), THIS_IMAGE(coarray), THIS_IMAGE(coarray, dim)
intrin_13.7.172.f90	LCOBOUND(coarray) and LCOBOUND(coarray,dim)
intrin_13.7.79.f90	IMAGE_INDEX(coarray, subs)
intrin_13.7.91.f90	UCOBOUND(COARRAY[, DIM, KIND])
intrin_6.7.3.2.11.f90	ALLOCATE and DEALLOCATE act as barriers
item_4.8.a.f90	Subobjects of a coarray is also a coarray
pointer_4.5.4.6b.f90	association of pointer components of coarrays with local objects
intrin_8.5.7d.f90	STOP and LOCK construct with STAT=STAT_LOCKED specifier
intrin_8.5.7e.f90	STOP and LOCK construct with STAT=STAT_LOCKED_OTHER_IMAGE specifier
intrin_8.5.7f.f90	STOP and LOCK construct with STAT=STAT_UNLOCKED specifier
derived_4.5.4.f90	(non-)coarray COMPONENTS of (non-)coarray derived types

Table 3. Confidence Test files in the UH - CAF Validation Tests suite

File	Description
atomic_8.5.2.f90	Atomic subroutines
critical_8.1.5.f90	CRITICAL - END CRITICAL sections
intrin_8.5.6.f90	LOCK & UNLOCK without STAT specifier
sync_8.5.3.f90	SYNC ALL without STAT specifier
sync_8.5.4a.f90	SYNC IMAGES(arr) paired with SYNC IMAGES(*)
sync_8.5.4b.f90	call to SYNC IMAGES(arr), should not behave like SYNC ALL

Table 4. Fault (tolerance) Test files in the UH - CAF Validation Tests suite

File	Description
sync_8.5.7a.f90	STOP and SYNC ALL with STAT=STAT_STOPPED_IMAGE specifier
sync_8.5.7b.f90	STOP and SYNC IMAGES(arr) with STAT=STAT_STOPPED_IMAGE specifier
sync_8.5.7c.f90	STOP and SYNC IMAGES(*) with STAT=STAT_STOPPED_IMAGE specifier

4 Test results

Tables 5,6, and 7 list the extent of support of coarrays in different CAF compiler implementations as detected by the UH CAF validation test suite:

Table 5. Results of Feature tests

DESCRIPTION	OpenUH	Intel	G95	Cray
CHARACTER coarrays	YES	YES	YES	YES
similar translation of co-subscripts and subscripts	YES	YES	YES	YES
reference of coarray without [ ] implies local object	YES	YES	YES	YES
attribute CODIMENSION + remote accesses at single integer/real boundary	YES	Exec times out	Exec fails	YES
(non-)coarray COMPONENTS of (non-)coarray derived types	YES	YES	YES	YES
explicit shape, assumed size, assumed shape, allocatable dummy arguments	YES	YES	comp fails	YES
NUM_IMAGES() returns the number of images launched	YES	YES	YES	YES
THIS_IMAGE(), THIS_IMAGE(coarray), THIS_IMAGE(coarray, dim)	YES	YES	YES	YES
LCOBOUND(coarray) and LCOBOUND(coarray,dim)	YES	YES	YES	YES
IMAGE_INDEX(coarray, subs)	YES	YES	YES	YES
UCOBOUND(COARRAY[, DIM, KIND])	YES	YES	YES	YES
ALLOCATE and DEALLOCATE act as barriers	YES	YES	YES	YES
STOP and LOCK construct with STAT=STAT_LOCKED specifier	YES	YES	comp fails	Exec fails
STOP and LOCK construct with STAT=STAT_LOCKED_OTHER_IMAGE specifier	Exec times out	Exec times out	comp fails	Exec times out
STOP and LOCK construct with STAT=STAT_UNLOCKED specifier	YES	Exec times out	comp fails	Exec fails
subobjects if a coarray is also a coarray	YES	YES	YES	YES
association of pointer components of coarrays with local objects	YES	Exec times out	YES	YES

Table 6. Results of Confidence tests

DESCRIPTION	OpenUH	Intel	G95	Cray
Atomic subroutines	Passes with 0% confidence	Fails comp	comp fails	Passes with 4% confidence
CRITICAL - END CRITICAL sections	YES	Passes with 0% confidence	YES	Exec fails
LOCK & UNLOCK without STAT specifier	YES	Fails comp	comp fails	YES
SYNC ALL without STAT specifier	YES	Passes with 0% confidence	YES	YES
SYNC IMAGES(arr) paired with SYNC IMAGES(*)	YES	Exec fails	YES	YES
call to SYNC IMAGES(arr), should not behave like SYNC ALL	YES	Exec fails	YES	YES

Table 7. Results of Fault tests

DESCRIPTION	OpenUH	Intel	G95	Cray
STOP and SYNC ALL with STAT=STAT_STOPPED_IMAGE specifier	YES	Exec fails	Exec fails	Exec fails
STOP and SYNC IMAGES(arr) with STAT=STAT_STOPPED_IMAGE specifier	YES	YES	Exec fails	YES
STOP and SYNC IMAGES(*) with STAT=STAT_STOPPED_IMAGE specifier	YES	Exec fails	Exec fails	YES

References

1. ISO: "international standard ISO/IEC 1539-1:2010 (E) - Draft (for Ballot)", Third edition (June)
2. Numrich, R.W., Reid, J.: Co-arrays in the next fortran standard. SIGPLAN Fortran Forum 24(2), 4–17 (August 2005), http://doi.acm.org/10.1145/1080399.1080400