```
**************************************************
                  J3/02-231

Date:     July 15 2002
To:       J3
From:     Aleksandar Donev
Subject:  Enhanced C_LOC: Null Arguments
Reference: Paper J3/02-229
**************************************************
```

Summary

I propose a modification of the specification of C_LOC from ISO_C_BINDING to allow null dynamic arguments, i.e. disassociated scalar pointers and unallocated allocatables, in which case it will return C_NULL_PTR. This allows for simpler interfacing with C.

I hope the modification will be discussed at meeting 162, which I will attend.

Motivation

J3's 007/R2 only allows allocated allocatable arguments of interoperable type and type parameters. Assuming my proposal in 02-229 is accepted, associated scalar pointers of interoperable type and type parameters will also be accepted. I believe we should augment the wording to also allow an unallocated or diassociated argument X as well, in which case it will be specified that C_LOC returns a null C pointer (C_NULL_PTR).

This allows for more natural, more consistent and simpler interfacing with C, where a null pointer commonly signals what in Fortran would be an unallocated or nullified pointer. For example, the MPI C binding has many dummy arguments which will commonly be null or simply ignored, such as all receiving buffers on processors which do not receive in the given communication operation. See the attached example at the end of this paper.

In fact, most, if not all compilers, will not check an argument to C_LOC for its association or allocation status, but rather simply pass on the same address that they use internally to implement the dynamic Fortran variable. In all implementations I know, these are already null C pointers for disassociated/unallocated variables. Therefore most implementations will already comply with my proposal, even though the standard will demand users to write extra run-time checks to insure full compliance. I also believe most users will not notice this as their programs will work fine, and also be simpler!

Example

For example, take the following (simplified) Scatter MPI-like operation:

```
void Gather(void* sndbuf, void* rcvbuf, int count)
```

with the Fortran binding:

```
INTERFACE
   SUBROUTINE Gather(send_buffer, receive_buffer, send_count), BIND(C)
      USE ISO_C_BINDING
      TYPE(C_PTR), VALUE :: send_buffer
         ! The host gathers send_count elements from each
         ! processor from send_buffer into the receive_buffer
      TYPE(C_PTR), VALUE :: receive_buffer
         ! Only meaningful on the root (host) processor
         ! It is ignored on others!
      INTEGER(C_INT), VALUE :: send_count
         ! How many elements to send to the host
   END SUBROUTINE
END INTERFACE Gather
```

```
REAL(C_DOUBLE), DIMENSION(:), ALLOCATABLE, TARGET :: receive_buffer, send_buffer
TYPE(C_PTR) :: receive_buffer_address
! ... Other variables
```

At present, the Fortran code to do a Gather would be:

```
IF(am_host) THEN
   ALLOCATE(receive_buffer(send_count*n_processors))
   receive_buffer_address=C_LOC(receive_buffer)
ELSE
   receive_buffer_address=C_NULL_PTR
END IF
CALL Gather(C_LOC(send_buffer), receive_buffer_address, send_count)
```

Where as with the proposed modification it will simply be:

```
IF(am_host) ALLOCATE(receive_buffer(send_count*n_processors))
CALL Gather(C_LOC(send_buffer), C_LOC(receive_buffer), send_count)
```