|epcc|

Message Passing Programming: Array Issues in Fortran

David Henty

There is slightly subtle issue with mixing Fortran 90 array syntax with non-blocking MPI communications that I would like to explain.

First, consider the following code:

```
allocate(buf(n))
call MPI_Issend(buf, n, ....)
deallocate(buf)
```

This is not a correct piece of code because there is no guarantee that the `Issend` will have completed before the deallocation of `buf`. If the communication hasn't taken place by this point, then when the message is actually sent the `buf` array won't exist any longer and the program might crash or you could get unpredictable behaviour. It is only safe to deallocate the memory after the matching Wait.

Note that this isn't specific to Fortran – you could do exactly the same thing in C using `malloc` and `free`. However, the problem arises in a less obvious way in Fortran due to its more sophisticated array handling.

Now consider the following code:

```
real, dimension(m,n) :: array
...
call MPI_Issend(array(1,1:n), n, MPI_REAL, ...)
```

This isn't obviously wrong, but actually has the same problem as the first piece of code. The reason is that the section of the array we are passing is not contiguous in memory, whereas a Fortran subroutine assumes that all its array arguments are contiguous. What the compiler actually does is something like:

```
allocate buf(n)
buf(1:n) = array(1,1:n)
call MPI_Issend(buf, n, MPI_REAL, ...)
array(1,1:n) = buf(1:n)
deallocate(buf)
```

For any normal Fortran routine this is perfectly safe and happens completely transparently to the user. Unfortunately, non-blocking MPI operations are NOT normal routines as they continue to operate AFTER the function call has returned.

For this reason it is not safe to pass Fortran array subsections to any non-blocking MPI operations (the same problem happens with `Irecv`). You are probably safe for contiguous array

subsections as a good compiler will spot that the data is already contiguous and will not create an explicit temporary array. However, this can't be guaranteed.

The advice is:

> For contiguous array sections, pass the starting array element rather than an explicit subsection, i.e. do NOT do:
>
> ```
> call MPI_Issend(array(1:m,1), m, ...)
> ```
>
> You should do:
>
> ```
> call MPI_Issend(array(1,1), m, ...)
> ```
>
> For non-contiguous sections, you should do one of the following:
>
> - define an appropriate vector derived type for the columns (e.g. called `coltype`) and do something like:
>
>   ```
>   call MPI_Issend(array(1,1), 1, colytpe, ...)
>   ```
>
> - do an explicit copy in and copy out to a contiguous temporary buffer, ensuring that the buffer stays in existence until after the non-blocking communication is guaranteed to have finished (of course you can also only do the copy out when the communication is guaranteed to have finished).

Note that this is often said to be a problem specific to Fortran. It is NOT specific to Fortran – it is just that the problem arises in a more subtle way in Fortran because of the way that array syntax has to be handled. However, you can still get the problem in C (and Fortran) in other non-obvious ways.

For example, the following piece of C code has exactly the same issues:

```
#define n 100

void mysend(float array[n])
{
  int i;
  float buf[n];

  for (i=0; i < n; i++)
  {
    buf[i] = array[i];
  }

  MPI_Issend(buf, n, MPI_FLOAT, ....);
}
```

because the temporary storage for `buf` is local to `mysend` and may be deallocated by the compiler immediately it returns from `mysend`.