

# Fortran90 Bindings for Charm++\*

February 12, 2012

Charm++ is a parallel object language based on C++. The `f90charm` module is to provide Fortran90 programs a `f90` interface to Charm++. Using `F90Charm` interface, users can write Fortran90 programs in a fashion similar to Charm++, which allows creation of parallel object arrays (Chare Arrays) and sending messages between them.

To interface Fortran90 to Charm++ and thus obtain a parallel version of your program you need to do the following things:

1. Write a Charm Interface file (extension `.ci`)
2. Write your F90 program with `f90charmmain()` as main program;
3. Write implementation of Chare entry functions in `f90` program;
4. Compile and Link with Charm's Fortran library
5. Run it !

## 1 Overview

Here I suppose you've already known most concepts in `charm++` and done some `charm++` programming.

Unlike in C++, we don't have class here in Fortran90. Thus, Chare is represented as a fortran type structure. Here is an example:

```
## Just replace Hello throughout with your chare's name. ##
## and add your chare's personal data below where indicated ##
## Everything else remains the same ##
```

---

\*last modified 4/1/2001 by Gengbin Zheng

```

MODULE HelloMod

  TYPE Hello
  ## your chare's data goes here, the integer below is an example ##
  integer data
  END TYPE

  TYPE HelloPtr
  TYPE (Hello), POINTER :: obj
  integer*8 aid
  END TYPE

END MODULE

```

You can think of this module as a chare declaration. Type [Hello] defines arbitrary user program data and HelloPtr defines the Chare pointer which fortran program will use later to communicate with the f90charm runtime library, the [aid] is the handle of array returned by f90charm library, user shouldn't change it..

As same in C++ charm program, you need to write a .ci interface file so that charm translator will generate helper functions. The syntax of .ci files are the same as in Charm++, however, for Fortran90 charm, there are certain constraints. First, you don't need to declare the main chare as like in Charm++; Second, it currently only support up to 3D Chare array, you cannot define Chare and Group types. Third, there is no message declaration in .ci files, all the entry functions must be declared in the parameter marshelling fashion as in Charm++. So, what you can do in the .ci files is to define readonly variables and 1-3D chare arrays with parameter marshelling entry functions.

It is programmer's responsibility to write the implementation of chare's entry functions. The decl and def files generated by Charm++ translator define the interface functions programmer need to write.

For each Chare defined in the .ci file, user must write these functions for charm++ f90 runtime:

```
SUBROUTINE <ChareName>_allocate(objPtr, aid, index)
```

You can think of this function as a constructor for each array element with array index [index]. For 3D array, for example, replace index in the example by 3D array index [index1, index2, index3]. In this function user must allocate memory for the Chare's user data and perform initialization.

For each chare entry method you declared, you should write the corresponding fortran90 subroutine for it:

```
SUBROUTINE entry(charePtr, myIndex, data1, data2 ... )
```

Note, the first argument is the Chare pointer as you declared previously, the second argument is the array index which will be passed from charm runtime. The rest of the parameters should be the same as you declare the entry function in .ci files. For higher dimensional arrays, replace myIndex by "myIndex1, myIndex2" for example.

On the other side, the decl/def files generated by Charm++ translator also provide these functions for Chare creation and remote method invocation. for each Chare declared in .ci files, these subroutines are generated for use in Fortran90 program:

```
<ChareName>_CkNew(integer n, integer*8 aid)
```

This subroutine creates the chare array of size n. For higher dimensional array creation, specify one integer for each dimension. For example, to create a 3D array:

```
<ChareName>_CkNew(integer dim1, integer dim2, integer dim3, integer*8 aid)
```

And for each entry method, this function is available for use in f90 program if it is 1D array:

```
SendTo_<ChareName>_<Entry>(charePtr, myIndex, data1, data2 ... )
```

This subroutine will send a message to the array element with the index as myIndex. Similarly for arrays with higher dimensions, replace myIndex by corresponding number of array indices.

There are several others things you need to know.

First, as same in Charm++, each .ci file will generate two header files: .decl.h and .def.h. However, in Fortran90 charm, you are not able to include these C++ files in Fortran90 code. Thus, currently, it is user's task to write a simple C++ code to include these two headers files. You should also declare readonly variables as in C++ in this file. This is as simple as this:

```
#include "hello.decl.h"
int chunkSize; // readonly variables define here
#include "hello.def.h"
```

In future, this file can be generated automatically by translator.

Second, you can still use readonly variables as in Charm++. However, since there is no global variables as in C++ in fortran90, you have to access them explicitly via function call. Here are the two helper functions that translator generates:

take the readonly variable chunkSize as an example,

```
Set_Chunksize(chunkSize);
Get_Chunksize(chunkSize);
```

These two functions can be used in user's fortran program to set and get readonly variables.

Third, for user's convenience, several charm++ runtime library functions have their Fortran interface defined in f90charm library. These currently include:

```
CkExit()
CkMyPe(integer mype)
CkNumPes(integer pes)
CkPrintf(...) // note, the format string must terminated with '$$'
```

Here is a summary of current constraints to write f90 binding charm++ programs:

1. in .ci files, only 1-3D Chare array is supported.
2. readonly variables must be basic types, ie. they have to be integer, float, etc scalar types or array types of these basic scalar types.
3. instead of program main, your f90 main program starts from subroutine f90charmmain.

All these are best explained with an example: the hello program. It is a simple ring program. When executed, an array of several parallel Chares is created. Each chare "says" hello when it receives a message, and then sends a message to the next chare. The Fortran f90charmmain() subroutine starts off the events. And the SayHi() subroutine does the say-hello and call next chare to forward.

## 2 Writing Charm++ Interface File

In this step, you need to write a Charm++ interface file with extension of .ci. In this file you can declare parallel Chare Arrays and their entry functions. The syntax is same as in Charm++.

```
## Just replace Hello throughout with your chare's name. ##
## and add your chare's entry points below where indicated ##
## Everything else remains the same ##
mainmodule hello {
```

```

// declare readonly variables which once set is available to all
// Chares across processors.
readonly int chunkSize;

array [1D] Hello {
  entry Hello();

  // Note how your Fortran function takes the above defined
  // message instead of a list of parameters.
  entry void SayHi(int a, double b, int n, int arr[n]);

  // Other entry points go here

};
};

```

Note, you cannot declare main chare in the interface file, you also are not supposed to declare messages. Furthermore, the entry functions must be declared with explicit parameters instead of using messages.

### 3 Writing F90 Program

To start, you need to create a Fortran Module to represent a chare, e.g. {ChareName}Mod.

```

## Just replace Hello throughout with your chare's name. ##
## and add your chare's personal data below where indicated ##
## Everything else remains the same ##
MODULE HelloMod

TYPE Hello
## your chare's data goes here ##
integer data
END TYPE

TYPE HelloPtr
TYPE (Hello), POINTER :: obj
integer*8 aid
END TYPE

```

```
END MODULE
```

In the Fortran file you must write an allocate function for this chare with the name: Hello\_allocate.

```
## Just replace Hello throughout with your chare's name. ##
## Everything else remains the same ##
SUBROUTINE Hello_allocate(objPtr, aid, index)
USE HelloMod
TYPE(HelloPtr) objPtr
integer*8 aid
integer index

allocate(objPtr%obj)
objPtr%aid = aid;
## you can initialize the Chare user data here
objPtr%obj%data = index
END SUBROUTINE
```

Now that you have the chare and the chare constructor function, you can start to write one or more entry functions as declared in .ci files.

```
## p1, p2, etc represent user parameters
## the "objPtr, myIndex" stuff is required in every Entry Point.
## CkExit() must be called by the chare to terminate.
SUBROUTINE SayHi(objPtr, myIndex, data, data2, len, s)
USE HelloMod
IMPLICIT NONE

TYPE(HelloPtr) objPtr
integer myIndex
integer data
double precision data2
integer len
integer s(len)

objPtr%obj%data = 20
if (myIndex < 4) then
    call SendTo_Hello_SayHi(objPtr%aid, myIndex+1, 1, data2, len, s);
else
```

```
        call CkExit()
    endif
```

Now, you can write the main program to create the chare array and start the program by sending the first message.

```

SUBROUTINE f90charmain()
USE HelloMod
integer i
double precision d
integer*8 aid
integer s(8)

call Hello_CkNew(5, aid)

call set_ChunkSize(10);

do i=1,8
    s(i) = i;
enddo
d = 2.50
call SendTo_Hello_SayHi(aid, 0, 1, d, 4, s(3:6));

END
```

This main program creates an chare array Hello of size 5 and send a message with an integer, an double and array of integers to the array element of index 0.

## 4 Compile and Link

Lastly, you need to compile and link the Fortran program with the Charm program as follows: (Let's say you have written `hellof.f90`, `hello.ci` and `hello.C`)

```
charmc hello.ci -language f90charm
```

will create `hello.decl.h`, `hello.def.h`

```
charmc -c hello.C
```

will compile the `hello.C` with `hello.decl.h`, `hello.def.h`.

```
charmc -c hellof.f90
```

charmc will invoke fortran compiler;

```
charmc -o hello hello.o hellof.o -language f90charm
```

will link hellof.o, hello.o against Charm's Fortran90 library to create your executable program 'hello'.

There is a 2D array example at [charm/examples/charm++/f90charm/hello2D](#).

## 5 Run Program

To run the program, type:

```
./charmrun +p2 hello
```

which will run 'hello' on two virtual processors.