The Interface-Translator Version of the Charm++ Programming Language Manual

August 18, 1997

The Interface-Translator version of Charm++ (ITC++) was developed at the Parallel Programming Laboratory of the University of Illinois at Urbana-Champaign. ITC++ was coded by Jayant DeSouza with much initial support from Sanjeev Krishnan. Thread support was implemented by Laxmikant Kale. The interface was designed mainly by Sanjeev and Laxmikant Kale with contributions from Jayant and Robert Brunner. Robert also provided vital testing of the features of ITC++. This document was created by Jayant and is based on Sanjeev's specification of the interface.

Contents

1 Introduction **3**

2 Structure **4**

2.1 Basic Files and Compilation 4

2.2 One Module, One or Multiple Source Code Files 4

2.3 Multiple Modules 5

3 Syntax **6**

3.1 Source Code (.C) AND Interface File (.ci) Changes 6

3.1.1 Supported Features of Charm++ 6

3.1.2 Currently unsupported Features of Charm++ 7

3.1.3 Extensions to Charm++: EXTERN, THREADED and Return Values 7

3.2 Source code (.C) Changes 7

1 Introduction

CHARM++ is an object oriented portable parallel programming language based on C++. Its program structure and execution model, language constructs and runtime system calls are described[1] in the Charm++ manual.

This is the documentation for the Interface Translator (IT a.k.a non-translator or NT) version of Charm++. Using IT Charm++ involves changing your Charm++ source code (.C) files, and creating an interface file (.ci) for each module. This document (1) gives instructions for compiling IT Charm++programs, and (2) lists the syntax of each Charm++ construct in the original (Charm++) language, and the matching construct in the IT Charm++code (.C) as well as what needs to correspondingly appear in the interface file (.ci).

[1]For a description of the underlying design philosophy please refer to the following papers :
L. V. Kale and Sanjeev Krishnan, *"CHARM++ : Parallel Programming with Message-Driven Objects"*, in "Parallel Programming Using C++", MIT Press, 1995. (to be published).
L. V. Kale and Sanjeev Krishnan, *"CHARM++ : A Portable Concurrent Object Oriented System Based On C++"*, Proceedings of the Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA), September 1993.

2 Structure of an IT Charm++ Program

This section describes what files are needed and created when using IT Charm++, and how an IT Charm++ program is compiled.

2.1 Basic Files and Compilation

Users of the non-translator version must #include "ckdefs.h" (which contains macros provided by IT Charm++) in their C++ code (pgm.C). Users must create an additional interface file (pgm.ci) containing the essential components of their program as described below. The interface file must be named <module>.ci, where <module> is the name of the module, and the .ci stands for Charm++ Interface file.

Compile the IT Charm++program pgm.C as usual, ensuring that the pgm.ci file precedes the pgm.C file:

```
charmc -language charm++ -o pgm pgm.ci pgm.C
                OR
charmc -language charm++ pgm.ci
charmc -language charm++ -o pgm pgm.C
```

The interface file, pgm.ci, is transparently processed by a tool called charmxi (Charm++ interface translator) to produce a "pgm.top.h" file and a "pgm.bot.h" file. (The non-parsing translator will NOT look at or cross-check with pgm.h and pgm.C files; it will SOLELY process the pgm.ci file.) For each module only one pgm.top.h and one pgm.bot.h file will be produced from the pgm.ci file.

The resulting pgm.top.h file contains only declarations, not definitions; and the pgm.bot.h file contains functions and variable definitions generated by the interface translator.

Generally speaking, the pgm.top.h file must be #include'd at the top of pgm.C, and the pgm.bot.h must be #include'd at the bottom of pgm.C BY THE USER. Here are more details about this:

2.2 One Module, One or Multiple Source Code Files

1. There is only one module.ci file
2. There may be several .C and .h files containing the code for the module.
3. In all the .C files of a module, the user #include's ckdefs.h and module.top.h at the beginning of the file as follows:

```
#include "ckdefs.h"
#include "chare.h"
#include "c++interface.h"
#include "module.top.h"
```

4. In only ONE .C file, the user #include's module.bot.h at the end of the file. This .C file MUST contain the class declaration, so the wrapper functions don't give an error.

2.3 Multiple Modules

1. There is ONE module.ci file for EACH module.
2. Sharing of user-defined declarations/definitions across modules is done with the usual C++ extern declarations, in the HEADER .h files.
3. IN ADDITION, if moduleAA is using a message or chare (i.e. some construct) which is defined in moduleBB, moduleAA needs to declare as extern the accessed message or chare in the moduleAA.ci file. This is done by preceding the construct with the word 'extern' in the interface file.

Compile the .ci files before any Charm++ program file:

```
charmc -language charm++ -c moduleAA.ci moduleBB.ci
charmc -language charm++ -c moduleAA.C
charmc -language charm++ -c moduleBB.C
charmc -language charm++ -o pgm moduleAA.o moduleBB.o
```

3 Syntax

Some changes to the Charm++ syntax affect both the .C and the .ci files; others affect only .C files.

3.1 Source Code (.C) AND Interface File (.ci) Changes

3.1.1 Supported Features of Charm++

The following five rules represent how declarations need to be changed in the source code files and what needs to appear in the interface file:

```
Original:       represents the construct in the regular Charm++ pgm.C file
IT .C:  represents the same construct in the IT Charm++ pgm.C file
Interface:      indicates how the construct is declared in the pgm.ci file
```

1. Messages

   ```
   Original:       message class MsgType { ... };
   IT .C:  class MsgType : public comm_object { ... };
   Interface:      message MsgType;
   ```

   If the message has pack and unpack functions, then the interface file should have:

   ```
                   packmessage MsgType;
   ```

2. Chares

   ```
   Original:       chare class ChareType {
                       entry:
                           void EP1(MsgType *m) { ... }
                   };
   IT .C:  class ChareType : public chare {
                       public:
                           void EP1(MsgType *m) { ... }
                   };
   Interface:      chare ChareType {
                           entry EP1(MsgType *);
                   };
   ```

3. Branch Office Chares

   ```
   Original:       chare class BOCType : public groupmember {
                       entry:
                           void EP1(MsgType *m) { ... }
                   };
   ```

```
IT .C:  class BOCType : public groupmember {
                public:
                    void EP1(MsgType *m) { ... }
              };
Interface:    group BOCType {
                    entry EP1(MsgType *);
              };
```

4. Readonly Variables and Messages

```
Original:     readonly int x;
IT .C:  int x;
Interface:    readonly int x;
```

If the type is "MsgType *" (instead of "int" as above), then it is a readonly message, else a readonly variable.

5. Tables

```
Original:     table tblname;
IT .C:  table tblname;
Interface:    table tblname;
```

3.1.2 Currently unsupported Features of Charm++

The non-translator version does not support accumulators, monotonics (they can be implemented as BOC libraries), or varsize messages (they can be implemented by the user using pack/unpack functions).

3.1.3 Extensions to Charm++: EXTERN, THREADED and Return Values

- To declare a chare or message as extern: This is done by preceding the construct with the word 'extern' in the interface file.
- To declare an entry point as a threaded entry point: This is done by preceding the construct with the word 'threaded' in the interface file.
- For an entry point to return a message of type MsgType: This is indicated by preceding the entry point with the MsgType in the interface file.

If more than one of the above words must precede a construct, the words must appear in the order listed.

3.2 Source code (.C) Changes

The following are purely changes to the Charm++ system calls (nothing is needed to be done in the interface file). Basically, since IT Charm++ does not parse the .C files, the user needs to perform the syntax changes that the original Charm++ did. The macros are provided in ckdefs.h

```
. Direct syntax changes
      ORIGINAL                IT .C
    ChareType handle    --->    ChareIdType
    ChareType group     --->    ChareGroupType
    writeonce handle    --->    WriteOnceID

. Remote and Asynchronous Method invocations: % => stuff :
      ORIGINAL                IT .C
    ChareId=>EP(msg)    --->    CSendMsg(ChareType,EP,msg,ChareId);
    BOCId[Pe]=>EP(msg)  --->    CSendMsgBranch(BOCType,EP,msg,BOCId,Pe);
    BOCId[ALL]=>EP(msg) --->    CBroadcastMsgBranch(BOCType,EP,msg,BOCId);

. Local Method invocation for groupmembers (Branch Office Chares)
    BocId[LOCAL]->Fn()  --->    CLocalBranch(BocType, BocId)->Fn()

. writeonceid->DerefWriteOnce(...)  --->  DerefWriteOnce(writeonceid, ...)

. &(ChareType::EP)     --->    GetEntryPtr(ChareType,EP)

. new MessageType      --->    new (MsgIndex(MessageType)) MessageType

. new (prio) MessageType  --->  new (MsgIndex(MessageType), prio) MessageType

. newchare ChareType(msg)  ---> newchare(ChareType, msg)

. newchare (vid, pe) ChareType  ---> newchare2(ChareType, msg, vid, pe)

. newgroup ChareType(msg)  ---> newgroup(ChareType, msg)

. newgroup (returnEP, returnID) ChareType  ---> newgroup2(ChareType, msg, returnEP, returnID)
```