Charm/Charm++/Converse Installation and Usage

Parallel Programming Laboratory
Department of Computer Science
University of Illinois at Urbana–Champaign

August 18, 1997

1 Introduction

In this manual, we describe how to install Charm, Charm++ and Converse, how to compile and execute programs, and the available command line options. We also describe various queueing and load balancing strategies, and the various modes of execution of Charm and Charm++ programs. Details on the Interface Translator version of Charm++ are in the appendix.

2 Installing Converse, Charm, and Charm++

The three programming systems are distributed as a single package — one always installs all three. They will all be installed in a single directory which, for lack of a better term, we will call the "charm" directory. It would be typical to choose `/usr/local/charm` as the location for the charm directory, although any location will do. Our explanation of the installation process will assume that `/usr/local/charm` is to be the location of the charm directory, if not, you must mentally translate.

Download the tar-file file appropriate to your machine. For example, if you are installing the version of charm for networks of Sun workstations running SunOS 4.X, download `net-sun.tar.Z`:

```
% ftp a.cs.uiuc.edu
Connected to a.cs.uiuc.edu.
220 a.cs.uiuc.edu FTP server ready.
Name: anonymous
331 Guest login ok, send your complete e-mail address as password.
Password: jyelon@cs.uiuc.edu
230-
230-    Welcome to the University of Illinois at Urbana-Champaign,
230-                 Dept. of Computer Science FTP server.
230-
230 Guest login ok, access restrictions apply.
ftp> cd pub/research-groups/CHARM/CHARM.4.5
250 CWD command successful.
ftp> binary
200 Type set to I.
ftp> get net-sun.tar.gz
200 PORT command successful.
150 Opening BINARY mode data connection for net-sun.tar.gz (648211 bytes).
226 Transfer complete.
local: net-sun.tar.gz remote: net-sun.tar.gz
648211 bytes received in 38 seconds (17 Kbytes/s)
ftp> quit
```

Now that you have downloaded it, you are ready to install it: uncompress it, untar it, and move the directory thereby created to `/usr/local/charm`:

```
% gunzip net-sun.tar.gz
% tar xf net-sun.tar
% mv net-sun /usr/local/charm
```

You are almost ready to go. The last thing you need to do is put the charm binaries in your path. There are two acceptable ways to do this. The first is to simply reset your path:

```
% setenv PATH /usr/local/charm/bin:$PATH
```

The second is to install `charmc` in a common directory available to all users of your machine, such as `/usr/local/bin`. *DO NOT* move or copy `charmc`: If you do, all three programming systems will cease to function correctly. The only reliable way to put `charmc` in a common directory is to symbolically link it:

```
% ln -s /usr/local/charm/bin/charmc /usr/local/bin/charmc
```

The three programming systems, converse, charm, and charm++, are probably ready to go. However, there is one possible situation that they may not be. If your parallel machine has a separate "host" computer on which you do compilation, and if your "host" computer is of a type different from the one we anticipated, then some of the binaries may be compiled for the wrong kind of CPU. To correct this situation, you may need to rebuild the charm and charm++ compilers:

```
% cd /usr/local/charm/src
% make all
```

This should complete the installation process. You are ready to test converse, charm, and charm++. However, there are a few things to attend to before you start using them on a regular basis. These are security issues and disk space consumption.

2.1 Security Issues

On most computers, charm programs are simple binaries, and they pose no more security issues than any other program would. The exceptions to this rule are the network versions `net-sun`, `net-sol`, `net-hp`, and `net-rs6k`. This section only applies to users of the networked versions, all other users may skip it.

The network versions utilize many unix processes communicating with each other via UDP. No attempt is currently being made to filter out unauthorized packets. Therefore, it is theoretically possible to mount a security attack by sending UDP packets to an executing converse, charm, or charm++ program's sockets.

The second security issue associated with networked programs is associated with the fact that we, the charm developers, need evidence that our tools are being used. (Such evidence is useful in convincing funding agencies to continue to support our work.) To this end, we have inserted code in the network `conv-host` program (described later) to notify us that our software is being used. Implementationally speaking, the `conv-host` program sends a single `UDP` packet to `charm.cs.uiuc.edu`. This data is put to one use only: it is gathered into tables recording the internet domains in which our software is being used, the number of individuals at each internet domain, and the frequency with which it is used.

We recognize that some users may have objections to our notification code. Therefore, we have provided a second copy of the `conv-host` program with the notification code removed. If you look within the charm `bin` directory, you will find these programs:

```
% cd /usr/local/charm/bin
% ls conv-host*
```

```
conv-host
conv-host.notify
conv-host.silent
```

The program `conv-host.silent` has the notification code removed. To permanently deactivate notification, you may use the version without the notification code:

```
% cd /usr/local/charm/bin
% cp conv-host.silent conv-host
```

Although versions for some other machines (eg. ncube) contain programs named `conv-host.notify` and `conv-host.silent`, they never actually notify us. The existence of the extra files is just to make our compilation scripts more consistent across versions. The *only* versions that ever notify us are the network versions.

2.2 Reducing disk usage

This section describes how you may delete parts of the distribution to save disk space.

The charm directory contains a collection of example-programs and test-programs. These may be deleted with no other effects:

```
% rm -r /usr/local/charm/pgms
```

The source code for the translators may be deleted, if you know the binaries are operational:

```
% rm -r /usr/local/charm/src
```

If you are not using charm++, you may delete the charm++ compiler:

```
% rm /usr/local/charm/bin/charmxlat++
% rm /usr/local/charm/bin/charmfilter++
```

Similarly, if you are not using charm, you may delete the charm compiler:

```
% rm /usr/local/charm/bin/charmxlat
% rm /usr/local/charm/bin/charmfilter
```

If you are using neither charm nor charm++, you may delete the combined charm/charm++ runtime library:

```
% rm /usr/local/charm/lib/libck-*
% rm /usr/local/charm/lib/libcharm.a
```

If you are using neither charm nor charm++, you may delete the charm and charm++ include files. Everything in `include` which doesn't start with `conv` (eg, converse) is charm-related:

```
% cd /usr/local/charm/include
% mkdir save
% mv conv* save
% rm *.h *.int
% mv save/* .
% rmdir save
```

You may delete the programs `conv-host.notify` and `conv-host.silent`, although please read the section on security first.

Finally, you may `strip` all the binaries in `/usr/local/charm/bin`, if we have not already done so.

2.3 Using more than one version of Charm on the same Machine

It is common to wish to install more than one version of charm on the same machine. For example, we often use the uniprocessor version for debugging our programs (it supports `gdb` in a simple way), and then we switch to the networked version to run our programs in parallel.

To do this, you will need more than one charm directory. For example, we use `/usr/local/charm.uth` and `/usr/local/charm.net`. Simply install the versions independently, one at a time. All charm directories should be in the same parent directory.

Each version contains the `charmc` script. Which `charmc` you use to compile your programs determines which behavior the program will exhibit. For example, if you install a uniprocessor version in `/usr/local/charm.uth` and a networked version in `/usr/local/charm.net`, and if you compile using `/usr/local/charm.uth/bin/charmc`, then your programs will be uniprocessor programs.

The exception to this rule is if you use the `-machine` option of `charmc`. `-machine` causes charm to look for another installed charm directory with the specified name. For example, if you installed `/usr/local/charm.uth` and `/usr/local/charm.net` as described above, you could specify:

```
% /usr/local/charm.net/bin/charmc -machine charm.uth myprog.p
```

Even though you are running the `charmc` script from the network version, it will produce uniprocessor binaries, since you told it explicitly to use that version.

3 Compiling Converse, Charm, and Charm++ Programs

The charmc program standardizes compiling and linking procedures among various machines and operating systems. The word "charmc" is slightly misleading, this is a general-purpose tool for compiling and linking, not restricted to charm programs at all.

Charmc can perform the following tasks. The (simplified) syntax for each of these modes is shown. Caution: in reality, one almost always has to add some command-line options in addition to the simplified syntax shown below. The options are described next.

```
 * Compile C                     charmc -o pgm.o pgm.c
 * Compile C++                   charmc -o pgm.o pgm.C
 * Compile Charm                 charmc -o pgm.o pgm.p
 * Compile old-style Charm++     charmc -o pgm.o pgm.P
 * Link                          charmc -o pgm   obj1.o obj2.o obj3.o...
 * Compile + Link                charmc -o pgm   src1.c src2.C src3.p src4.P
 * Create Library                charmc -o lib.a obj1.o obj2.o obj3.o...
 * CPM preprocessing             charmc -gen-cpm file.c
 * Preprocess modern Charm++     charmc file.ci
```

Charmc has been given data on how to invoke the compilers on each different platform. This data is in a file named `conv-mach.csh`, which can be found in the same directory where `charmc` resides. Local modifications to the commands or options `charmc` uses may usually be accomplished by editing `conv-mach.csh`.

Charmc automatically figures out where the charm lib and include directories are — at no point do you have to configure this information. However, the code that finds the lib and include directories can be confused if you remove charmc from its normal directory, or rearrange the directory tree. Thus, the files in the charm distribution must be left where they are, relative to each other. Use symbolic links if you want to put a copy of charmc into a local bin directory.

The following command-line options are available to users of charmc:

`-o` *output-file*: Output file name. Note: charmc only ever produces one output file at a time. Because of this, you cannot compile multiple source files at once, unless you then link or archive them into a single output-file. If exactly one source-file is specified, then an output file will be selected by default using the obvious rule (eg, if the input file if pgm.c, the output file is pgm.o). If multiple input files are specified, you must manually specify the name of the output file, which must be a library or executable.

`-c`: Ignored. There for compatibility with `cc`.

`-D*`: Defines preprocessor variables from the command line at compile time.

`-I`: Add a directory to the search path for preprocessor include files.

`-g`: Causes compiled files to include debugging information.

`-L*`: Add a directory to the search path for libraries selected by the `-l` command.

`-l*`: Specifies libraries to link in.

`-O`: Causes files to be compiled with maximum optimization.

`-NO`: If this follows -O on the command line, it turns optimization back off. This is just a convenience for simple-minded makefiles.

`-s`: Strip the executable of debugging symbols. Only meaningful when producing an executable.

`-save`: Intermediate files produced by the Charm or Charm++ translator are saved.

`-verbose`: All commands executed by charmc are echoed to stdout.

`-seq`: Indicates that we're compiling sequential code. On parallel machines with front ends, this option also means that the code is for the front end. This option is only valid with C and C++ files.

`-machine` *machine-type*: If more than one version of converse/charm/charm++ has been installed, this option allows selection of versions other than the default. The default *machine-type* is the version in which the `charmc` being run resides. See the previous chapter on installing charm.

`-use-fastest-cc`: Some environments provide more than one C compiler (cc and gcc, for example). Usually, charmc prefers the less buggy of the two. This option causes charmc to switch to the most aggressive compiler, regardless of whether it's buggy or not.

`-use-reliable-cc`: Some environments provide more than one C compiler (cc and gcc, for example). Usually, charmc prefers the less buggy of the two, but not always. This option causes charmc to switch to the most reliable compiler, regardless of whether it produces slow code or not.

`-language {converse|converse++|charm|charm++}`: When linking with charmc, one must specify the "language". This is just a way to help charmc include the right libraries. Pick the "language" according to this table:

- **Charm++** if your program includes Charm++, Charm, C++, and C.
- **Charm** if your program includes Charm and C.
- **Converse++** if your program includes C++ and C.
- **Converse** if your program includes only C.

`-balance` *load-balance-strategy*: When compiling Charm or Charm++, one must include a load-balancing library. There are currently three to choose from: `rand`, `acwn`, and `mngr` are supported. Default is `-balance rand`.

`-tracemode` *tracing-mode*: Selects the desired degree of tracing for Charm and Charm++ programs. See the Charm manual and the Projections and SummaryTool manuals for more information. Currently supported modes are `none`, `summary`, and `projections`. Default is `-tracemode none`.

`-c++` *C++ compiler*: Forces the specified C++ compiler to be used.

`-cc` *C-compiler*: Forces the specified C compiler to be used.

`-cp` *copy-file*: Creates a copy of the output file in *copy-file*.

`-cpp-option` *options*: Options passed to the C pre-processor.

`-ld` *linker*: Use this option only when compiling programs that do not include C++ modules. Forces charmc to use the specified linker.

`-ld++` *linker*: Use this option only when compiling programs that include C++ modules. Forces charmc to use the specified linker.

`-ld++-option` *options*: Options passed to the linker for `-language charm++`.

`-ld-option` *options*: Options passed to the linker for `-language charm`.

`-ldro-option` *options*: Options passes to the linker when linking `.o` files.

`-queue` *queueing strategy*: Currently ignored.

4 Executing Converse/Charm/Charm++ Programs

The Charm linker produces one executable file. On machines with a host (such as a network of workstations), a link to the proper host program **conv-host** is created in the user program directory. Sample execution examples are given below (the executable is called pgm). Exact details will differ from site to site. The list of Charm command line options is in Section 4.2.

- <u>**CM-5:**</u>

  pgm

  will run program pgm interactively from one of the partition managers. For programs with large resource requirements, use the utilities **jrun** and **jstat**. See the man pages at your site for more information. Also, the **+pN** option is recognized and runs **pgm** on **N** processors.

- <u>**nCUBE/2:**</u>

  xnc -d4 pgm

  runs pgm on a hypercube of dimension 4 (i.e. on 16 processors).

- <u>**Paragon running SunMos:**</u>

  yod -sz 4 pgm

  runs pgm on four processors.

- <u>**Paragon running OSF:**</u>

  pexec pgm -sz 4

  runs pgm on four processors.

- <u>**Network of workstations:**</u>

  conv-host pgm +p4

  executes pgm on 4 nodes. In a network environment, Charm must be able to locate the directory of the executable. If all workstations share a common file name space this is trivial. If they don't, Charm will attempt to find the executable in a directory with the same path from the **$HOME** directory. Pathname resolution is performed as follows:

  1. The system computes the absolute path of **pgm**.
  2. If the absolute path starts with the equivalent of **$HOME** or the current working directory, the beginning part of the path is replaced with the environment variable **$HOME** or the current working directory. However, if exec_home is specified in the nodes file (see below), the beginning part of the path is replaced with exec_home.
  3. The system tries to locate this program (with modified pathname and appended extension if specified) on all nodes.

The list of nodes must be specified in a file. The format of this file allows you to define groups of machines, giving each group a name. Each line of the nodes file is a command. The most important command is:

```
host <hostname>
```

which specifies a host. The other commands are qualifiers: they modify the properties of all hosts that follow them. The qualifiers are:

```
group <groupname>  - subsequent hosts are members of specified group
login <login>      - subsequent hosts use the specified login
passwd <passwd>    - subsequent hosts use the specified password
setup <cmd>        - subsequent hosts should execute cmd
home <dir>         - subsequent hosts should find programs under dir
cpus <n>           - subsequent hosts should use N light-weight processes
speed <s>          - subsequent hosts have relative speed rating
ext <extn>         - subsequent hosts should append extn to the pgm name
```

All of these accept "*" as an argument, this resets the modifier to its default value. Note that currently, the passwd, cpus, and speed factors are ignored. Inline qualifiers will be added:

```
host beauty ++cpus 2
```

and also user-defined qualifiers.

Here is a simple nodes file:

```
group kale-sun

host charm.cs.uiuc.edu
host dp.cs.uiuc.edu
host grace.cs.uiuc.edu
host dagger.cs.uiuc.edu

group kale-sol

host beauty.cs.uiuc.edu

group main

host localhost
```

This defines three groups of machines: group kale-sun, group kale-sol, and group main. The ++nodegroup option is used to specify which group of machines to use. Note that there is wraparound: if you specify more nodes than there are hosts in the group, it will reuse hosts. Thus,

```
conv-host pgm ++nodegroup kale-sun +p6
```

uses hosts (charm, dp, grace, dagger, charm, dp) respectively as nodes (0, 1, 2, 3, 4, 5).

If you don't specify a ++nodegroup, the default is ++nodegroup main. Thus, if one specifies

```
conv-host pgm +p4
```

it will use "localhost" four times. "localhost" is a Unix trick; it always find a name for whatever machine you're on.

Since the new nodes file is incompatible with the old nodes file, it has been renamed. It now is called ".nodelist", and all the options and environment variables pertaining to it have also been renamed `nodelist`.

The user is required to set up remote login permissions on all nodes using the .rhosts file in the home directory.

Note that the Charm linker will provide the correct executable. The user, however, needs to know how programs are run for the particular machine.

4.1 Running with the simulator

Converse provides a simple parallel machine simulator for developing and debugging purposes. It simulates a message passing system composed of a collection of processing nodes connected with a communication network. Each node is composed of an application processor, local memory, and a communication coprocessor. The simulator is a beta version, and it is not yet proven that the simulator timers for performance measurements produce realistic results.

In order to run Charm and Charm++ programs with the simulator:

- link user program with `<machine>/lib/libck-unimain.o`
- prepare a configuration file as described below
- to run, type `pgm +pN` (and possibly other runtime options) where N is the number of processors.

Currently only Charm and Charm++ programs can take advantage of the simulator features, because the `libck-unimain.o` file is Charm specific. In the future, a method to allow any Converse based program to use the simulator features will be devised.

The basic task of the simulator is to manage the message passing obeying various machine and network parameters. A message experiences delays in various components of the machine. These include: 1) sender application processor, 2) sender communication coprocesssor, 3) network, 4) receiver communication processor, and 5) receiver application processor. Each component of the delayed is modelled by the widely used formula $\alpha + n\beta$ where α is the startup cost, and β is the cost per byte. In addition to message delay parameters, there are others related to the network capacity and random variations in network delays. These parameters are specified in a configuration file named "sim.param" in the directory of the user program. If the simulator can't find this file, it assumes default values (mostly zero latencies). Figure 1 lists a sample configuration. The lines starting with the # sign are treated as comments. Each line contains a keyword followed by some numbers. The explanation of each keyword is given below:

`cpu_recv_cost` α and β values for the software cost of a message-receive at the application processor.

`cpu_send_cost` α and β values for the software cost of a message-send at the application processor.

`rcp_cost` α and β values for a message-receive at the communication processor.

`scp_cost` α and β values for a message-send at the communication processor.

`net_cost` α and β values for a message-send in the netowrk.

`cpu_queue_threshold_number` max number of messages queued at the application processors's incoming message queue.

`cpu_queue_threshold_size` max cumulative size of messages in bytes queued at the application processors's incoming message queue.

`cpu_queue_threshold_number` max number of messages in the incoming message queue of communication processor.

`rcp_queue_threshold_number` max number of messages in the incoming-message-queue of communication processors.

`rcp_queue_threshold_size` max cumulative size of messages in bytes in the incoming-message-queue of communication processors.

`net_queue_threshold_number` max number of transient messages in the network.

`net_queue_threshold_size` max cumulative size of transient messages in bytes in the network.

`latency-fixed` no random variations in the network latency (α)

`latency-rand` network latency (α) is incremented by a random value distributed exponentially. The first number after the keyword is the mean of the exponential distribution. The second number is the initial seed vbalue for the random number generator.

`processor_scale` The simulator scales the measured time execution of code-blocks by this value.

`periodic_interval` Converse has periodic checks for various purposes. This is the time on seconds those checks are called.

4.2 Command Line Options

A Charm program accepts the following command line options:

```
#latency parameters
cpu_recv_cost 1E-6 1E-7
cpu_send_cost 1E-6 1E-7
rcp_cost      1E-3 1E-7
scp_cost      1E-6 1E-7
net_cost      1E-6 1E-7


#capacity parameters
# choose one
cpu_nolimit
#cpu_queue_threshold_number 100000
#cpu_queue_threshold_size   100000


#choose one
scp_nolimit
#scp_queue_threshold_number 100000
#scp_queue_threshold_size   100000

#choose one
rcp_net_nolimit
#rcp_queue_threshold_number 100000
#rcp_queue_threshold_size   100000
#net_queue_threshold_number 100000
#net_queue_threshold_size   100000

#random variations in latency
#choose one
latency-fixed
#latency-rand   0.0001 123456

processor_scale 1.0
periodic_interval 0.1
```

Figure 1: A sample configuration file for the simulator

+pN Run the program with N processors. The default is 1. Note that on some nonshared memory machines, e.g., nCUBE/2, the user must specify the number of processors using the command provided for that machine (e.g. **xnc** on the nCUBE/2). In such cases the **+p** option is ignored.

+ss Print summary statistics about chare creation. This option prints the total number of chare creation requests, and the total number of chare creation requests processed across all processors.

+cs Print statistics about the number of create chare messages requested and processed, the number of messages for chares requested and processed, and the number of messages for branch office chares requested and processed, on a per processor basis. Note that the number of messages created and processed for a particular type of message on a given node may not be the same, since a message may be processed by a different processor from the one originating the request.

user_options Options that are be interpreted by the user program may be included after all the system options. However, **user_options** cannot start with +. The **user_options** will be passed as arguments to the user program via the usual **argc/argv** construct to the **main** entry point of the main chare. Charm system options will not appear in **argc/argv**.

4.2.1 Additional Uniprocessor Command Line Options

The uniprocessor versions can be used to simulate multiple processors on a single workstation. Any number of processors between 1 and 32 can be simulated by using the **+p** option, limited only by the available memory on the uniprocessor workstation. By default, the uniprocessor versions handle a single message from each processor, going in order from processor 0 thru $P-1$ (where P is the number of processors) repeatedly.

4.2.2 Additional Network Command Line Options

The following ++ command line options are available in the network version:

++debug Run each node under gdb in an xterm window, prompting the user to begin execution.

++debug-no-pause Run each node under gdb in an xterm window immediately (i.e. without prompting the user to begin execution).

++maxrsh Maximum number of **rsh**'s to run at a time.

++resend-wait Timeout before retransmitting datagrams (in msec).

++resend-fail Timeout before retransmission fails (in msec). This parameter can help the user kill "runaway" processes, which may not be killed otherwise when the user interrupts the program before it completes execution. Currently a bug exists in the network version that may cause programs to terminate prematurely if this value is set too low and **scanf** operations are being performed.

++nodesfile File containing list of nodes.

If using the **++debug** option, the user must ensure the following:

1. **xterm**, **xdpyinfo**, and **gdb** must be in the user's path.
2. The path must be set in the **.cshrc** file, not the **.login** file, because **rsh** does not run the **.login** file.
3. The nodes must be authorized to create windows on the host machine (see man pages for **xhost** and **xauth**).

A Interface Translator

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

A.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:

A.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.

A.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
```

A.4 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).

A.5 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.