Parallel Programming Laboratory
University of Illinois at Urbana-Champaign

Charm++/Converse
Installation and Usage

Converse Parallel Programming Environment was developed as a group effort at Parallel Programming Laboratory, University of Illinois at Urbana-Champaign. The team consisted of Attila Gursoy, Sanjeev Krishnan, Joshua Yelon, Milind Bhandarkar, Narain Jagathesan, Robert Brunner and Laxmikant Kale. The most recent version of Converse has had inputs from Milind Bhandarkar, Laxmikant Kale, Robert Brunner, Terry Wilmarth, Parthasarathy Ramachandran, Krishnan Varadarajan, and Jeffrey Wright.

Version 5.4 (Release 1)

University of Illinois
Charm++/Converse Parallel Programming System Software
Non-Exclusive, Non-Commercial Use License

Contact Information:
The best contact path for licensing issues is by e-mail to `kale@cs.uiuc.edu` or send correspondence to:

Prof. L. V. Kale
Dept. of Computer Science
University of Illinois
201 N. Goodwin Ave
Urbana, Illinois 61801 USA
FAX: (217) 333-3501

Contents

1 Introduction **3**

2 Installing Charm++ **4**
2.1 Security Issues 4
2.2 Reducing disk usage 5

3 Compiling Charm++ Programs **6**

4 Executing Charm++ Programs **8**
4.1 Command Line Options 8
4.1.1 Additional Network Options 8
4.1.2 Multicore Options 10
4.2 Nodelist file 10
4.2.1 IO buffering options 11

1 Introduction

In this manual, we describe how to download and install the Charm++ parallel language and runtime system. We also describe how to compile and run Charm++ programs.

2 Installing Charm++

You can install Charm++ as either source code or a precompiled binary package. Downloading source code is more flexible, since you can choose the options you want; but a precompiled binary is slightly easier to get running.

You begin by downloading Charm++ from our web site: http://charm.cs.uiuc.edu/download.html

Unpack Charm++ using a tool capable of extracting gzip'd tar files, such as tar (on Unix) or WinZIP (under Windows). Charm++ will be extracted to a directory called "charm". If you choose the source distribution, read the included "charm/README" file for detailed instructions on building Charm++ from source.

The main directories in a Charm++ installation are:

charm/bin Executables, such as charmc and charmrun, used by Charm++.

charm/doc Documentation for Charm++, such as this document. Distributed as LaTeX source code; HTML and PDF versions can be built or downloaded from our web site.

charm/include The Charm++ C++ and Fortran user include files (.h).

charm/lib The libraries (.a) that comprise Charm++.

charm/pgms Example Charm++ programs.

charm/src Source code for Charm++ itself.

charm/tmp Directory where Charm++ is built.

charm/tools Visualization tools for Charm++ programs.

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 only exception is the network version `net-*`, which has the following issues.

The network versions utilize many unix processes communicating with each other via UDP. Only a simple attempt is currently made to filter out unauthorized packets. Therefore, it is theoretically possible to mount a security attack by sending UDP packets to an executing Converse 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 `charmrun` program (described later) to notify us that our software is being used. This notification is a single `UDP` packet sent by `charmrun` 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 `charmrun` program with the notification code removed. If you look within the charm `bin` directory, you will find these programs:

```
% cd charm/bin
% ls charmrun*
charmrun
charmrun-notify
charmrun-silent
```

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

```
% cd charm/bin
% cp charmrun.silent charmrun
```

The *only* versions of Charm++ 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 charm/pgms
```

You may also `strip` all the binaries in `charm/bin`.

3 Compiling Charm++ Programs

The `charmc` program, located in "charm/bin", standardizes compiling and linking procedures among various machines and operating systems. "charmc" is a general-purpose tool for compiling and linking, not only restricted to CHARM++ programs.

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
 * Link                                 charmc -o pgm   obj1.o obj2.o obj3.o...
 * Compile + Link                       charmc -o pgm   src1.c src2.ci src3.C
 * Create Library                       charmc -o lib.a obj1.o obj2.o obj3.o...
 * CPM preprocessing                    charmc -gen-cpm file.c
 * Translate Charm++ Interface File charmc file.ci
```

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/bin, charm/include, and charm/lib directories must be left where they are relative to each other.

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`*`symbol`*`[=`*`value`*`]`: 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.

`-module` *`m1`*`[,`*`m2`*`[,...]]` Specifies additional CHARM++ modules to link in. Similar to `-l`, but also registers CHARM++ parallel objects. See the library's documentation for whether to use `-l` or `-module`.

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

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

`-production`: Enable architecture-specific production-mode features. For instance, use available hardware features more aggressively. It's probably a bad idea to build some objects with this, and others without.

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

`-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.

`-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|charm++|sdag|ampi|fem|f90charm`}**:** 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++, C++, and C.
- **Converse** if your program includes C or C++.
- **sdag** if your program includes structured dagger.
- **f90charm** if your program includes f90 Charm interface.

`-balance` ***seed load-balance-strategy*****:** When linking any Converse program (including any Charm++ or sdag program), one must include a seed load-balancing library. There are currently three to choose from: `rand`, `test`, and `neighbor` are supported. Default is `-balance rand`.

When linking with `neighbor` seed load balancer, one can also specify a virtual tolpogy for constructing neighbors during run-time using `+LBTopo topo`, where *topo* can be one of (a) ring, (b) mesh2d, (c) mesh3d and (d) graph. The default is mesh2d.

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

`-memory` ***memory-mode*****:** Selects the implementation of `malloc` and `free` to use. Select a memory mode from the table below.

- **os** Use the operating system's standard memory routines.
- **gnu** Use a set of GNU memory routines.
- **paranoid** Use an error-checking set of routines. These routines will detect common mistakes such as buffer overruns, underruns, double-deletes, and use-after-delete. The extra checks slow down programs, so this version should not be used in production code.
- **verbose** Use a tracing set of memory routines. Every memory-related call results in a line printed to standard out. This version is useful for detecting memory leaks.
- **default** Use the default, which depends on the version of Charm++.

`-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 converse`.

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

4 Executing Charm++ Programs

When compiling CHARM++ programs, the charmc linker produces both an executable file and a program called `charmrun`, which is used to load the executable onto the parallel machine.

To run a CHARM++ program named "pgm" on four processors, type:

```
charmrun pgm +p4
```

Programs built using the network version of CHARM++ can be run alone, without charmrun. This restricts you to using the processors on the local machine, but it is convenient and often useful for debugging. For example, a CHARM++ program can be run on one processor in the debugger using:

```
gdb pgm
```

If the program needs some environment variables to be set for its execution on compute nodes (such as library paths), they can be set in .charmrunrc under home directory. charmrun will run that shell script before running the executable.

4.1 Command Line Options

A CHARM++ program accepts the following command line options:

`+pN` Run the program with N processors. The default is 1.

`+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 mixed with 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`.

4.1.1 Additional Network Options

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

`++local` Run charm program only on local machines. No remote shell invocation is needed in this case. It starts node programs right on your local machine. This could be useful if you just want to run small program on only one machine, for example, your laptop.

`++mpiexec` Use the cluster's `mpiexec` job launcher instead of the built in rsh/ssh method.

This will pass `-n $P` to indicate how many processes to launch. An executable named something other than `mpiexec` can be used with the additional argument `++remote-shell` *runmpi*, with 'runmpi' replaced by the necessary name.

Use of this option can potentially provide a few benefits:

- Faster startup compared to the SSH/RSH approach charmrun would otherwise use
- No need to generate a nodelist file
- Multi-node job startup on clusters that do not allow connections from the head/login nodes to the compute nodes

At present, this option depends on the environment variables for some common MPI implementations. It supports OpenMPI (`OMPI_COMM_WORLD_RANK` and `OMPI_COMM_WORLD_SIZE`) and M(VA)PICH (`MPIRUN_RANK` and `MPIRUN_NPROCS` or `PMI_RANK` and `PMI_SIZE`).

`++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).

If using one of the `++debug` or `++debug-no-pause` options, the user must ensure the following:

1. The `DISPLAY` environment variable points to your terminal. SSH's X11 forwarding does not work properly with CHARM++.
2. The nodes must be authorized to create windows on the host machine (see man pages for `xhost` and `xauth`).
3. `xterm`, `xdpyinfo`, and `gdb` must be in the user's path.
4. The path must be set in the `.cshrc` file, not the `.login` file, because `rsh` does not run the `.login` file.

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

`++nodelist` File containing list of nodes.

`++ppn` number of pes per node

`++help` print help messages

`++runscript` script to run node-program with

`++xterm` which xterm to use

`++in-xterm` Run each node in an xterm window

`++display` X Display for xterm

`++debugger` which debugger to use

`++remote-shell` which remote shell to use

`++useip` Use IP address provided for charmrun IP

`++usehostname` Send nodes our symbolic hostname instead of IP address

`++server-auth` CCS Authentication file

`++server-port` Port to listen for CCS requests

`++server` Enable client-server (CCS) mode

`++nodegroup` which group of nodes to use

`++verbose` Print diagnostic messages

`++timeout` seconds to wait per host connection

`++p` number of processes to create

4.1.2 Multicore Options

On multicore platforms, operating systems (by default) are free to move processes and threads among cores to balance load. This however sometimes can degrade the performance of Charm++ applications due to the extra overhead of moving processes and threads, especailly when Charm++ applications has already implemented its own dynamic load balancing.

Charm++ provides the following runtime options to set the processor affinity automatically so that processes or threads no longer move. When cpu affinity is supported by an operating system (tested at Charm++ configuration time), same runtime options can be used for all flavors of Charm++ versions including network and MPI versions, smp and non-smp versions.

`+setcpuaffinity` set cpu affinity automatically for processes (when Charm++ is based on non-smp versions) or threads (when smp)

`+excludecore <core #>` does not set cpu affinity for the given core number. One can use this option multiple times to provide a list of core numbers to avoid.

`+pemap L[-U[:S[.R]]][,...]` Bind the execution threads to the sequence of cores described by the arguments using the operating system's CPU affinity functions.

A single number identifies a particular core. Two numbers separated by a dash identify an inclusive range (*lower bound* and *upper bound*). If they are followed by a colon and another number (a *stride*), that range will be stepped through in increments of the additional number. Within each stride, a dot followed by a *run* will indicate how many cores to use from that starting point.

For example, the sequence `0-8:2,16,20-24` includes cores 0, 2, 4, 6, 8, 16, 20, 21, 22, 23, 24. On a 4-way quad-core system, if one wanted to use 3 cores from each socket, one could write this as `0-15:4.3`.

`+commap p[,q,...]` Bind communication threads to the listed cores, one per process.

4.2 Nodelist file

For network of workstations, the list of machines to run the program can be specified in a file. Without a nodelist file, Charm++ runs the program only on the local machine.

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> <qualifiers>
```

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
`shell <shell>`	- subsequent hosts use the specified remote shell
`setup <cmd>`	- subsequent hosts should execute cmd
`pathfix <dir1> <dir2>`	- subsequent hosts should replace dir1 with dir2 in the program path
`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

Note: By default, charmrun uses a remote shell "rsh" to spawn node processes on the remote hosts. The `shell` qualifier can be used to override it with say, "ssh". One can set the `CONV_RSH` environment variable or use charmrun option `++remote-shell` to override the default remote shell for all hosts with unspecified `shell` qualifier.

All qualifiers 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 are also allowed:

```
host beauty ++cpus 2 ++shell ssh
```

Except for "group", every other qualifier can be inlined, with the restriction that if the "setup" qualifier is inlined, it should be the last qualifier on the "host" or "group" statement line.

Here is a simple nodes file:

```
group kale-sun ++cpus 1
  host charm.cs.uiuc.edu ++shell ssh
  host dp.cs.uiuc.edu
  host grace.cs.uiuc.edu
  host dagger.cs.uiuc.edu
group kale-sol
  host beauty.cs.uiuc.edu ++cpus 2
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,

```
charmrun 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

```
charmrun pgm +p4
```

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

The user is required to set up remote login permissions on all nodes using the ".rhosts" file in the home directory if "rsh" is used for remote login into the hosts. If "ssh" is used, the user will have to setup password-less login to remote hosts using RSA authentication based on a key-pair and adding public keys to ".ssh/authorized_keys" file. See "ssh" documentation for more information.

In a network environment, `charmrun` 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, `charmrun` 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 `++pathfix dir1 dir2` is specified in the nodes file (see above), the part of the path matching `dir1` is replaced with `dir2`.
3. The system tries to locate this program (with modified pathname and appended extension if specified) on all nodes.

4.2.1 IO buffering options

There may be circumstances where a Charm++ application may want to take or relinquish control of stdout buffer flushing. Most systems default to giving the Charm++ runtime control over stdout but a few default to giving the application that control. The user can override these system defaults with the following runtime options:

`+io_flush_user` User (application) controls stdout flushing

`+io_flush_system` The Charm++ runtime controls flushing