Parallel Object Programming in C++

Getting Started

The POP-C++ Team
Grid & Cloud Computing Group
http://gridgroup.hefr.ch
popcinfo@hefr.ch

Hes·so // FRIBOURG FREIBURG
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences
Western Switzerland

Software Version : 2.5
Manual Version : 2.5-a

Parallel Object Programming in C++
Getting Started
Manual version : 2.5-a, October 2012

`http://gridgroup.hefr.ch`

This work was partially funded by the CoreGRID Network of
Excellence, in the European Commission's 6th Framework
Program and by the University of Applied Sciences of
Western Switzerland of Fribourg.

The POP-C++ Team :

Pierre Kuonen
Tuan Anh Nguyen
Jean-François Roche
Valentin Clément
David Zanella
Marcelo Pasin
Laurent Winkler
Nicolas Brasey

Problems or questions concerning POP-C++ can be submitted on the POP-C++ web site or be sent by e-mail to:
`popcinfo@hefr.ch`

Table of content

CHAPTER 1

Download POP-C++

1.1 Introduction to POP-C++

POP-C++ is a comprehensive object-oriented framework for developing efficient distributed applications in large distributed computing infrastructures such as Clusters, Grid, P2P or Clouds. It consists of a programming suite (language, compiler) and a run-time system.

The POP-C++ language is a minimal extension of the C++ programming language, that implements the parallel object model. The extension has been kept minimal in order to make the POP-C++ language as close as possible to C++, to ease learning of POP-C++ and to make existing C++ libraries and programs easy to port in POP-C++. POP-C++ is developed and maintained by the Grid & Cloud Computing Group of the University of Applied Science of Western Switzerland at Fribourg.

1.2 Latest stable version

The latest stable release of POP-C++ is version 2.5 (released in October 2012). We strongly recommend to use this version. Get the latest stable version of POP-C++ here:

- `http://gridgroup.hefr.ch/popc/doku.php/download`

There exists two version of POP-C++ software, one for linux systems (Ubuntu, Fedora, etc) and one for MacOS systems. Download the version which is appropriated to your owns system.

Important (If you have already installed an older version of POP-C++**) :**

- Note that the names of all global variables have could have changed from PAROC_… to POPC_… and the names of scripts from paroc_… to popc_…
- You need to regenerate your `object.map` files since the architecture notation has changed.
- You must recompile all your POP-C++ programs with the version 2.5.
- Do not forget to stop (or kill) the Job Manager and then to restart it after having finished the installation of the last version of POP-C++

1.3 Previous stable versions

The previous stable releases of POP-C++ are versions 2.0, 1.3, 1.1.1, 1.1 and 1.0. They are now outdated and **we strongly recommend to update to the new version.**

There also exists a “Cygwin version” to be able to run POP-C++ on Windows systems using Cygwin. **This release is based on version 1.1.1 and is not maintained anymore.**

Demo programs

A set of demo programs is available with the sources of POP-C++. You can find very simple examples on how to program in POP-C++ in `examples/` and more advanced demos in the `demos/` directory.

CHAPTER

2 POP-C++ Installation

2.1 Introduction

This tutorial explains how to install POP-C++ and compile a first program. For a more advanced installation, please refer to the chapter 5 of the POP-C++ user manual. If you have comments or difficulties with this installation contact the POP-C++ team at `popcinfo@hefr.ch`.

2.2 Before the installation

2.2.1 Requirements

POP-C++ runs on a standard Linux and on MacOS. The necessary packages are:

- the **g++** compiler
- the **Gnu Tar archiver**
- the **Gnu Make utility**
- the **library zlib** (development version)
- Other requirements (optional, only necessary if you want to modify the POP-C++ tool itself)
  - the **Gnu Bison** to regenerate the parser
  - **Autoconf, automake** and **libtool** to regenerate the configuration files.

If you are using `apt-get`, to install `zlib-devel` (for example on Ubuntu), use the following command line:

```
sudo apt-get install g++ zlib1g-dev
```

If you are NOT using `apt-get`, such as on MacOs systems, just go to internet and look for the `zlib` library for your OS and install the full package. One possible link to find `zlib` is the home page : `www.zlib.net/`.

2.3 Installation

2.3.1 Compiling

To compile and install POP-C++ enter the following commands.

```
tar -xzf popcpp_<version.subversion>.tar.gz
cd popcpp_<version.subversion>
./configure
make
make install
```

Note

- You must have full rights (read, write and execute) on the directories you compile and install POP-C++.

If you experience problem when compiling POP-C++ (`make` command), correct the problem and, before relaunching the compilation, do `make clean` to force a full recompilation of the POP-C++ package.

The default installation directory is **`/usr/local/popc`**, you can change it by using

```
./configure --prefix=<inst-dir>
```

2.3.2 Installation script

When you run **`make install`**, the installation script asks you how to setup the system. You have two options : a *simple (standalone)* or a *standard* installation. If you want to use POP-C++ only on one machine (standalone), select the *simple* installation. If you want to use POP-C++ on a network of machines (cluster, grid,...), you must select the *standard* version and parametrize POP-C++ for your environment. We recommend usage of the simple version for beginners abd this document only presents the simple installation.

Note

- You can, at anytime, switch from one installation to the other by parametrizing POP-C++. This can be done by using the `popc_setup` command.

When you launch the installation (`make install`) several technical messages are displayed and eventually the following text and question are displayed

```
 POP-C++ will now be configured. To do this, some information must be
 entered manually.

*************************************************************************
They are 2 different ways to to configure the POP-C++ Services :

- if you are new to POP-C++ and simply want to try it on your machine,
  select a simple installation (y). No further question will be asked.
```

```
- if you want to use POP-C++ on a grid using the jobmanager, or any other
advanced
  option, select a standard installation (n). More questions will be asked.

For more information about which installation suits you best
 please read the documentation.

If needed, you can change the configuration at any time by
executing '/usr/local/popc/sbin/popc_setup' or 'make install'

**************************************************************************

DO YOU WANT TO MAKE A SIMPLE INSTALLATION ? (y/n) :
y
```

Answer **y** to this question. The following message is displayed.

```
=======================================================
GENERATING SERVICE MAPS...
CONFIGURING POP-C++ SERVICES ON YOUR LOCAL MACHINE...
CONFIGURING THE RUNTIME ENVIRONMENT
SETTING UP RUNTIME ENVIRONMENT VARIABLES
=======================================================
CONFIGURING STARTUP SCRIPT FOR YOUR LOCAL MACHINE...
=======================================================
CONFIGURATION DONE!
=======================================================

IMPORTANT : Do not forget to add these lines to your .bashrc file or
equivalent :
---------
    POPC_LOCATION=/usr/local/popc
    export POPC_LOCATION=/usr/local/popc
    PATH=$PATH:$POPC_LOCATION/bin:$POPC_LOCATION/sbin

Press <Return> to continue
```

In the example shown above we assume that you are installing POP-C++ in the default directory i.e. `/usr/local/popc`, if not modify the `.bashrc` file (or equivalent) accordingly.

2.4 Environment

2.4.1 PATH

In order to have direct access to all POP-C++ commands you must add POP-C++ installation directory in your `PATH` environment variable. Add the following lines in your **`~/.bashrc`** file (or equivalent).

```
POPC_LOCATION=/usr/local/popc
export POPC_LOCATION=/usr/local/popc
PATH=$PATH:$POPC_LOCATION/bin:$POPC_LOCATION/sbin
```

Here we assume that you have installed POP-C++ is the directory: `/usr/local/popc`.

Note for MAC users

On MacOS the name of the file equivalent to **`~/.bashrc`** on Ubuntu is **`~/.profile`**

2.5 Test

A suite of test programs can be run to test the POP-C++ installation, but first you must launch the **job manager** and the **resource discovery service** (POP-C++ runtime services) by typing the following command:

```
SXXpopc start
```

You can find more information on starting and stopping the POP-C++ services in the section 5.5 of the POP-C++ User and Installation Manual.

Then launch the following script :

```
cd <popc-src>/test
./runtests -all
```

where `<popc-src>` is the directory you downloaded the source of POP-C++.

You also can run each test individually.

Type:

```
./runtests --help
```

for more information.

Note

- You must have full rights (read, write and execute) on the `<popc-src>` directory.

All tests must succeed.

In case of not understanding failures of execution, please report the problem to the POP-C++ team by sending an e-mail at: `popcinfo@hefr.ch`

CHAPTER

3 Introduction to programming in POP-C++

3.1 Introduction

In this section we will comment the source code files of the `myparclass` program to introduce to POP-C++ programming.

3.2 Files

3.2.1 MyParClass.ph

In POP-C++ `.ph` files contain headers (the declarations) of parallel classes. These files play the same role than `.h` files for C or for C++. Below is the content of the `MyParClass.ph` file which declares the `MyParClass` parallel class.

```
#ifndef MYPARCLASS_PH_
#define MYPARCLASS_PH_

parclass MyParClass
{
  classuid(1001);

  public:
    // constructor
    MyParClass()@{od.url("localhost");};

    // print the character c
    async seq void putchar(char c);

    // ask parallel object o to print the character c
    async seq void putchar(char c, MyParClass& o);
};
#endif
```

Compared to a normal `.h` file we can notice the following differences:

1. the keyword `class` is replaced by the key word `parclass`.
2. methods declarations are preceded by keywords `async` and `seq`.

3. there is a the directive `@{od.url("localhost");}` following the constructor.
4. the presence of the directive `classuid(1001)`. (see 3.3.2 in the user manual)

Parallel class

A parallel class is similar to a standard C++ class. It can have constructors, a destructor, methods and attributes. But unlike a C++ class, it is compiled as a separated executable file which can be executed and invoked remotely.

To declare a class as a parallel class we use the keyword **`parclass`**. Refer to section 3.2.1 of the user manual of POP-C++ for more details.

Method semantics

The methods have extra descriptors, such as **`sync/async`** or **`seq/conc/mutex`**. These descriptors allow to define the semantic of the methods i.e. the way the remote method is invoked. Refer to the section 3.2.3 of the POP-C++ user manual for a full description on the semantic of methods.

Object descriptor

You can note that constructors declaration have extra information. These are **object descriptors** specifying how and where parallel objects must be created. See section 3.2.4 of the POP-C++ user manual for more information on object descriptors.

3.2.2 MyParClass.cc

```
#include <stdio.h>
#include "myparclass.ph"

MyParClass::MyParClass(){}

void MyParClass::putchar(char c)
{
  printf("%c ", c);
}

void MyParClass::putchar(char c, MyParClass& o)
{
  o.putchar(c);
}

@pack(MyParClass);
```

The `MyParClass.cc` file contains the implementation of the class (also called: the body). It is very similar to a standard `.cc` file for a C++ program. Nevertheless, `.cc` files of a parallel classes must be terminated by the `pack` instruction.

```
@pack(POPCobject);
```

This instruction indicates to the POP-C++ compiler that this class is to be packed in one executable file. Refer to the section 3.3.1 of the POP-C++ user manual for detailed information on the `pack` directive.

3.2.3 main.cc

```
#include <stdio.h>
#include "MyParClass.ph"

int main(int argc, char** argv)

{
  printf("\nSTART program\n");

  MyParClass x[3];
  for (int i=0; i<10; i++) x[0].putchar('a', x[2]);
  for (int i=0; i<10; i++) x[1].putchar('b', x[2]);

  printf("\nEND of program\n\n");
  return 0;
}
```

The `main.cc` file syntax is absolutely similar to C++ code. You can note that parallel objects are created as standard C++ objects :

```
MyParClass x[3];
```

and that a method invocation is made as for C++ :

```
x[0].putchar('a', x[2]);
```

The POP-C++ compiler will create an executable which contains the main file. This is the file which will be launched on the local machine and will create the parallel objects on remote machines.

3.3 Compilation

3.3.1 The compilation process in POP-C++

To compile POP-C++ programs you must use the POP-C++ compiler. The name of this compiler is:

- `popcc`

The compilation is made in two phases. The first phase compiles the `main` program. This will allow to create the executable file which will be launched in the local machine when executing the program. In POP-C++ the `main` function is always executed on the local machine. In the second phase we compile the parallel classes to create the executable files for the remotes objects.

To compile an link the `main` of the example described in the previous section type:

- `popcc -o main main.cc myparclass.cc myparclass.ph`

This will create the `main` executable file you will use to launch your program. An important difference with standard C++ compilation is that, with POP-C++, you must also compile the headers of the parallel classes i.e. the `.ph` files (in our case the file `myparclass.ph`).

To compile the parallel class `MyParClass` type:

- `popcc -object myparclass.obj myparclass.cc myparclass.ph`

The option `-object` allows you to indicated the name of the file which will contains the executable of the parallel class. By convention we use the extension `.obj` for these files. Here again you have to compile the `.ph` file.

More information on the compilation process of POP-C++ can be found in the chapter 4 of the POP-C++ user manual.

3.1.2 The Map file

Before executing your program you need one more step to allow to indicate to the POP-C++ run-time where it can find the executable files to download on the remote machine when creating a parallel object (the `.obj` files). The list of these files have to be put in a file that you will give to the run-time when launching the execution.

Type the following command:

- `./myparclass -listlong > obj.map`

This will create the `obj.map file` which contains the necessary information for the POP-C++ run-time. More information on the Map file can be found in the section 4.2.3 of the POP-C++ user manual.

3.4 Running the program

To run POP-C++ program we use the command `popcrun`. This command has, at least, two parameters:

1. the name of the Map file
2. the name of the file which contains the executable code of the `main`

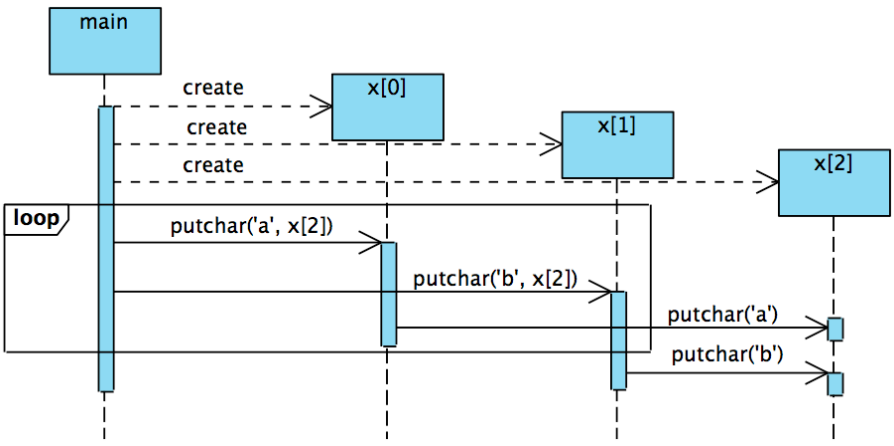
In our example we launch the execution with the following command:

- `popcrun obj.map ./main`

If the case the `main` program has parameters (also called: arguments) these parameters are added after `./main` as usual with C/C++ programs.

More information on running POP-C++ program can be found in the section 4.2.3 of the POP-C++ user manual.

Below is shown a sequence diagram which illustrates the execution of the `myparclass` program.



You can play by using different semantics for the methods of `MyParClass` and to observe the effect on the program execution.