How to compile in the Computers of the SGI-IZO

SGI-IZO

September 16, 2010

Abstract

In this article we describe how to compile and prepare the programs to run in the machines of the service. It also shows, with a simple example, how to easily submit and run your programs in any of all the computers of the service.

1 Preliminary facts

1.1 Do you have problems or you need help?

Contact the technicians.[1]

1.2 What is to compile?

The programs are usually written in a human readable language, in like Fortran or C emplyoing human understandable words such as copy, multiply, print, etc. Machines, however, work in binary language, they speak binary language. The compilation is the process in which a human readable code, e.g. Fortran, is translated to machine readable language, binary code.

The compiler is the program that make this translation, producing a binary for the machine in which it is run.[2]

1.3 About the architectures

In the SGI-IZO we have several processor types (architectures), namely ia64 (Itanium processors), x86_64 (Opteron, Xeon and core2duo processors) and x86 (Pentium IV, Pentium D . . . processors). The architectures define how the processor works, i.e., how it parses the instructions and the data. Usually, different architectures are not compatible. A program compiled to run in ia64 architecture does not work in a x86_64 architecture because it can not understand that binary code, ia64 instructions and x86_64 instructions have different implementation.

Itanium and x86_64 architectures are able to run x86 code. They have been built with backward compatibility but the performance may be poor, specially in the ia64 case.

Ideally, a program compiled in one architecture must run only in this architecture. Sometimes, unfortunatelly, this is not possible, and only then could be justified the usage of an x86 binary on the other architectures.

2 Compiling

2.1 The compiler

One of the most used compilers are the GNU ones, which are free software. Those are good compilers for x86 and x86_64 architectures, i.e., for Péndulo Grid and Opteron nodes in the

[1]`http://www.ehu.es/sgi/Personal_del_Servicio_tf.html`

[2]Cross-compiling over architectures is also possible.

Table 1: Commands to execute the different compilers.

Language	**GNU**	**Intel**
C	gcc	**icc**
C++	g++	**icpc**
fortran	gfortran	**ifort**

machines of the service. Nevertheless the GNU compilers, give very bad performance for the ia64 architecture, i.e., for the Itanium2 processor like in the Arina server and Itanium nodes.

In the service we have also installed the Intel compilers with usually have a very good performance, specially for Intel processors. This are the only recommended compilers for the ia64 architecture, Itanium2 nodes.[3]

In table 1 we summarize the commands to execute the different compilers. We will focus only in the Intel compilers. **We strongly recommend the use of the Intel compilers on the machines of the service.**

2.2 How to compile

In the rest of the document we will use the Intel Fortan compiler but everything is extensible to other language or compiler. As an example, we will a very simple fortran program that prints "Hello World" saved in a file with the name hola.f90:

```
program hola
write(*,*) "Hello world!"
end program hola
```

To compile this program type in the command line:

```
ifort hola.f90 -o hola
```

the "`-o hola`" option tells the program to rename the output, the binary will be created as `hola` and after compilation the `hola` binary file will appear in our directory. To execute it type:

```
./hola
```

and "`Hello world!`" will appear in the screen.

You can tune your compilation by using several flags. The most common one is `-O` which sets some optimizations. When a code is compiled with `Optimization` we ask the compiler to try to create a binary and modify it in order to be executed faster or more efficiently. By default, the optimization level is 2, we can increase it to 3, which is the maximum optimization level. For example:

```
ifort -O3 hola.f90 -o hola
```

During the optimization process, sometimes, the code is changed on a whay that it may produce erroneous results. Thus, **anytime that an optimization procedures is used when compiling, the correctness of the results should be checked with some tests**.

To learn more about different compiler options go to the manuals or use the `man` linux command, i.e. `man ifort`.

[3]The PGI compilers are available for the Opteron nodes but they will not be treated because it has note been upgraded in the last years.

Table 2: Returned value of the `arch` command for the different computing nodes

Node type	**main server**	`arch`
Itanium node in Arina cluster	Arina	ia64
Xeon node in Arina cluster	Guinness	x86_64
Opteron node in Arina cluster	Maiz	x86_64
PC node in Péndulo grid	Péndulo	i686

2.3 How to link libraries

There are very usefull mathematical libraries with very optimized codes for basic mathematical functions. You can check what libraries are installed in our machines in our web page.[4] For example, to use the `libfftw.a` library that performs Fast Fourier Transform that is installed on `/software/fftw-3.2.2/lib` directory you must use the `-L` option to tell the directory path and the `-l` option to pass the library name (without the `.a` suffix and without the `lib` prefix):

```
ifort hola.f90 -o hola -L/software/fftw-3.2.2/lib -lfftw
```

without the blank spaces after `-L` and `-l`. If several libraries are linked the order is important. In our the web (`www.ehu.es/sgi`) page usually there are examples about how to link the libraries.

3 Compiling a Code to be used in all the machines of the service

If you are going to use your own compiled code on the IZO-SGI computers, we encourage you to first analize the performance of the different architectutes, i.e. ia64 vs. x8_64, this might be very different. If so, you could focus your work on that specific computer type. If the performance is similar, then, we recommend you to use all the resources available at the service to improve your productivity.

The service has several architectures (ia64, x86 and x86_64), and you must run your programs in the architecture where they were compiled. If you compile your code only in the itanium server you **have to** be sure that it will only be run on itanium nodes. To do so, you **must** add the *itanium* label on the Torque script to the `nodes=` option when submitting the job.[5] In this way, you force the queue system to use **only** an itanium node.

The `qsub_all` command [6] allows you to submit a job to Arina and Péndulo at the same time, it will run in the first place it founds a free appropriate node (be aware of the walltime and memory limits of Péndulo[7]).

If you want to use all the computation nodes of the service you must compile your program for all the architectures and at run time select the proper binary for the architecture. This can be easilly done with the aid of the `arch` command that returns the architecture of the compute node. In table 2 you have the values that the command `arch` returns for the different architectures availabe at the IZO-SGI, and which is the server you should use to compile the programs for such architecture.

Appending the architecture value to the name of your binary, at the compilation time, is very helpfull to discriminate between de different architecures, and to make sutre that each binary runs in the proper architecture:

[4] `http://www.ehu.es/sgi/Librer_ias_tf.html`
[5] `http://www.ehu.es/sgi/Comandos_interes_tf.html`
[6] `http://www.ehu.es/sgi/Enviando_Arina_y_Pendulo_tf.html)`
[7] `http://www.ehu.es/sgi/GRID_Pendulo_tf.html`

```
ifort hola.f90 -o hola_ia64
```

and similarlly, for the the other architectures you have to generate the `hola_x86_64` and the `hola_i686` binaries. Then, if you set up your Torque scripts to execute the `hola` binary such as:

```
./hola_$(arch)
```

you will run the `hola_ia64` binary in itanium nodes, `hola_x86_64` in opteron and xeon nodes and `hola_i686` in Péndulo nodes, so you do not need to care about which binary type you should send to which specific node, this is done automatically by the computer.

3.1 A simple example:

In this subsection we describe with an example the steps to run your programs with `qsub_all` command in any of the nodes of the service, taking advantage of the `arch` command and allowing you to forget about which binary is executed where:

1. Connect to Arina.
2. Compile your program adding the architecture to the name
   ```
   ifort hola.f90 -o hola_ia64
   ```
3. Connect to Maiz.
4. Compile your program adding the architecture to the name.
   ```
   ifort hola.f90 -o hola_x86_64
   ```
5. Connect to Péndulo.
6. Compile your program adding the architecture to the name.
   ```
   ifort hola.f90 -o hola_i686
   ```
7. (optional) Make a `bin` directory in your `home` to store the binaries and avoid unnecessary copies. Copy or move there the binaries.
   ```
   mkdir $HOME/bin
   mv hola_* $HOME/bin
   ```
8. Edit your script to send jobs to torque and add this line to execute your new binaries
   ```
   $HOME/bin/hola_$(arch)
   ```
   or if you skip the previous step and your binaries are in the current directory
   `./hola_$(arch)`.
9. Submit it to Arina's and Péndulo's torque queue systems with the `qsub_all` command.

Now you can use `qstat_arina` and `qstat_pendulo` to check that you have one job in each server. The first that starts running will produce the results, the other will be automatically deleted.