Running the COSMO model on unusual hardware architectures

Davide Cesari

Arpae-SIMC, Bologna, Italy

1 Introduction

Nowadays more and more objects around us contain a computer, or simply *are* a computer, with a processor, some amount of memory and network capabilities, qualitatively analogous to those of a desktop computer or of an HPC (High Performance Computing) system. Moreover, many of these objects ("devices") are based on the Linux operating system and are capable to run executable codes generated by the gcc (GNU Compiler Collection), which includes the gfortran compiler that many of us use to compile the COSMO model. It is thus natural to ask oneself: could I run my favourite NWP model on such a device? The answer is yes, as it can be seen in the following sections.

2 Choice of the device

The most obvious choice of a device for this test would have been an Android smartphone, since these devices are nowadays rather powerful, universally widespread, and Android runs on top of a Linux kernel. However the commercial Android devices come with many barriers related to security and industrial secrets, so that it is possible to run arbitrary executables only after performing some hacking procedures (so-called "rooting" of the phone), which are risky and potentially illegal. For this reason a more open platform has been chosen: it is a fairly cheap (≈150 EUR) satellite receiver based on a MIPS processor and the Enigma2 TV interface on top of a basic Linux installation. The mass storage device consists in an external USB disk. The ultimate reason for this choice is simply that two of these devices are present in the author's home, see fig 1. An overview of the technical specifications of the devices is presented in table 1. The website of the manufacturer is http://www.gigablue.de.

Model	Processor	Memory
Gigablue 800 UE	333 Mhz Brcm4380	104 MB
Gigablue 800 SEplus	750 Mhz Broadcom BMIPS3300	222 MB

Table 1: Characteristics of the devices used for the test.

3 Preparing the model executable

Since this kind of devices has usually a limited amount of memory and of precompiled software packages available, it is hard to compile a complex Fortran code like the COSMO model on the device itself. Thus it has been decided to adopt the technique known as *cross-compiling*, which consists in generating the executable for the target architecture on a computer having a different architecture, reasonably a desktop computer. The Debian GNU/Linux distribution for traditional desktop computers is very suitable for this purpose, because it provides pre-compiled packages of gfortran compiler, with support for cross-compiling to many different target architectures

The instruction about how to setup a cross-compiling environment are described on a specific page of the Debian web site [1] and they can be resumed in four simple commands to be run by root on an installed system:

```
dpkg --add-architecture mipsel
apt-get update
apt-get install crossbuild-essential-mipsel
apt-get install gfortran-mipsel-linux-gnu
```

where `mipsel` indicates the MIPS architecture chosen for this particular test, so it may vary depending on the device chosen. After this step, a full set of compiling and linking commands becomes available on the system, simply by prefixing the usual command name (e.g. `gfortran`, `gcc`, `ar`) by the target-specific prefix, in this case `mipsel-linux-gnu-`.

For simplifying the task of compiling the COSMO model, it has been decided to build a sequential executable, without MPI and with only the DWD grib1 library, thus without the external libraries grib_api and netcdf. This step may require some editing of the COSMO code, depending on version, because such a combination is not well tested.

Moreover, a static executable has been built, in order to avoid problems related to possibly different system libraries between the cross-compiling system and the target system.

Thus, for performing cross-compilation, in the libgrib1 makefile and in COSMO `Fopts` file, the cross-compiling commands with the special prefix have been specified in place of the ordinary commands, such as mipsel-linux-gnu-gfortran for the Fortran compiler and linker. The COSMO makefile target used for this type of test has been `purseq`, i.e. sequential executable without assimilation and RTTOV code.

The flexible structure of the GNU gcc/gfortran compiler has proven to be very useful for this compilation task. The compiling process in gcc takes place in three main stages: a language-dependent and platform-independent front-end stage, which translates the program source code into an universal meta-language, a common optimisation stage and a final platform-dependent backend stage, where the object code is generated [2].

This structure allows to compile on different platforms with a minimum of changes in the command line and with access to almost the same set of optimisations.

4 The test case

Due to the small amount of memory available on the device used, it has been chosen to run a 3-dimensional test run on a very small grid. In order to make the test simpler and easily portable, the test has been configured to use artificial initial, boundary and external data.

The setup has been chosen among the test configurations available in the COSMO code and implemented by Ulrich Blahak [3]. The domain size used was $21 \times 21 \times 40$ grid points with 2km of grid step and 12s of time step. The initial state was characterized by a positive temperature anomaly in the center of the domain ("warm bubble").

Figure 1: Image of the Gigablue 800 UE used for one of the tests.

5 Performing the tests

Since the basic operating system in the devices under test is a plain GNU/Linux system, the procedure of running COSMO on this kind of system does not differ from what we are used to: opening a terminal on a personal computer, connecting to the device over the network with ssh and starting an executable from the command line. Unfortunately these particular devices do not have a console on the local TV screen, thus it is not possible neither to log in from the device itself, nor to have any feedback, on the TV screen, of the fact that a complex NWP model is running.

6 The results

The test case described above has been run on the two MIPS platforms described and, for comparison, on a state of the art HPC computing node (price ≈2000 EUR) using a single processing core.

The numerical results of the tests have shown only minor mutual discrepancies, within acceptable limits for this kind of simulations.

Table 2 summarises the results of the tests in terms of total wall-clock time required for one hour of forecast with the configuration described, as reported in the YUTIMING file.

Platform	wall clock time (s)
Gigablue 800 UE	1111
Gigablue 800 SEplus	28649
HPC computing node	12

Table 2: Summary of the tests performed.

The 800 SEplus model, although being more recent, has shown very poor performances because its processor lacks a FPU (Floating Point Unit), so the usual floating point operations with Fortran real numbers, which account for most of the COSMO model computing time, are emulated by the operating system using several integer operations per single floating point operation.

The 800 UE, equipped with a FPU, shows much better performances, although still being two orders of magnitude slower than a state of the art high performance computing platform. Even considering the smaller power consumption and price of the MIPS platform, the HPC node still wins in terms of performance per Watt and performance per price.

7 Conclusions

Since the computational resources required by the run of a state of the art operational NWP model are much higher than what can be provided even by a single node of an HPC system, it is clear that there is no use for such small devices for practically running a code like the COSMO model. And this is even more true when considering the relatively low parallel scalability potential of such devices.

However the feasibility of such an operation, through cross-compiling, has been proved and one key to the success has been the flexible structure of the free gcc/gfortran compiler.

Next step will be setting up cross-compilation with external dynamic libraries, including MPI, and trying a multiprocess run with MPI on a cheap multi-core minicomputer with arm architecture, like the Raspberry-Pi.

References

[1] CrossToolchains - Cross-compiling on Debian GNU/Linux.https://wiki.debian.org/CrossToolchains.

[2] GNU Compiler Collection, Wikipedia.https://en.wikipedia.org/wiki/GNU_Compiler_Collection

[3] Blahak, U., 2015: Simulating idealized cases with the COSMO-model. Available online.http://www.cosmo-model.org/content/model/documentation/core/artif_docu.pdf, **48**.