IS-ENES3 Milestone M8.1

A set of unit tests for XIOS

Reporting period: 01/01/2019 – 30/06/2020

Authors: Y. Wang, Y. Meurdesoif
Reviewer: G. Lister

Release date for review: 18/12/2019
Final date of issue: 10/01/2020

ABSTRACT

XIOS has been developed for several years and is widely used in climate models. However, the complexity of XIOS source code has given both users and developers more and more difficulties to debug and to maintain the consistency of XIOS. That is why we focused on our current work on the development of a set of unit tests and to check any development regressions automatically. In this milestone report, we present the automation procedure of compiling XIOS and the unit tests for checking XIOS functionalities. A web page is also developed for easy and direct view of the compiling status and test results of XIOS for the most recent revisions.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 824084

Table of contents

1. Objectives

Given the complexity of XIOS codes and its developments, it has become urgently important that the XIOS developers have a set of unit tests to ensure the non-regression of XIOS after each code modification. The main objective of the current work is to enable the automation of XIOS compilation and functionalities.
In another words, our work is to develop a mechanism to automatically launch the compile and then the set of unit tests of XIOS whenever a modification is applied on the XIOS source code. We also need to develop a simple web page to view the status of the compilation and the output of the unit tests. This web page will help us to easily and directly trace the development of XIOS. Also, we can spot any regression with the help of unit tests.

2. Description of work: Methodology and Results

We started by developing the set of unit tests. Each of these unit tests is targeted to check one given algorithm/functionality of XIOS. Each test has a reference result. When a unit test is launched after some source code modifications, the newly generated result will be compared with the reference. If no difference is shown, it means that no regression is observed and the new modification is validated. On the contrary, if the output result is different than the reference, we then have two possible conclusions. The first possibility is that the new modification of the XIOS source code introduced an error or a bug. The second possible situation is that the modification improved the performance or numerical accuracy of XIOS and the new output is valid and should be used to update the reference result. It is the XIOS developers' responsibility to investigate and to identify which conclusion should be kept.

What is worth mentioning is that these unit tests are flexible and configurable via external files. For example, we can test the zoom functionality of XIOS for the Gaussian grid and use the two-level-server mode by simply defining the grid type to be "Gaussian" and set the two-level-server to "True" in the configuration file of the zoom test. This mechanism is very useful especially when XIOS developers are asked to debug XIOS used by other users. Very often we encounter the situation where XIOS plugged within other models raises errors or has non-expected results. For most cases, it is difficult for an XIOS developer to reproduce the same error thus very time-consuming to debug. With the help of the unit tests, we can configure the external files to approach the target scenario without the knowledge of the model. Then we can work locally to find out whether or not the error or non-expected results come from XIOS and to find solutions if necessary.

In the second phase, we focused on the compilation of XIOS on different architectures. The idea is similar to that of the unit tests. We want to check if the building process of XIOS is consistent. This is essentially useful when using XIOS on a new architecture, switching to a different compiler, or building with different libraries. The building of XIOS is also configured by an external file in which we define the architecture used in XIOS "--arch" argument and the compile mode (production or debug mode). All compiler information along with the library modules is defined in the .arch file.

Two plain text reports are generated after the compile step and the unit tests. In the third phase of our work, we developed a web interface to view these reports. We developed some Python and JavaScript programs to translate the plain reports into JavaScript-type files that can be loaded by the main web page. The web page can be consulted via any web browser at the XIOS wiki page (https://forge.ipsl.jussieu.fr/ioserver/wiki) under the "XIOS Test Suite" tab[1]. For the moment, on the web page, we have two tables: one for the compile status of XIOS and one for the unit tests status of several revisions.

In the compile table (the upper table), each cell shows the building status of one revision of XIOS on a given machine (Irène and Jean-zay for the moment). Within the cell, the row label is the name of the "arch" and the column label the build mode. In the test table (the lower table), we show the overall status of the XIOS tests for a given revision and on a given machine (Irène and Jean-Zay for now). Each cell represents detailed information and can be revealed by clicking on the text.

With all three components (XIOS compilation, XIOS unit tests, and web interface) accomplished, we worked on the automation mechanism. First of all, all build and test reports are generated and stored on the machines locally. We need to make them visible through the web page. In other words,

1 https://vesg.ipsl.upmc.fr/thredds/fileServer/work/p86yann/XIOS_TEST_SUITE/xios_report.html

we have to lodge these JavaScript files on-line. On Irène, it is managed with the help of Thredds. On Jean-Zay, the files are lodged on the server of Forge.

The automation procedure is then controlled by Cron which is a time-based job scheduler in Unix-like computer operating systems. We use Cron to schedule the XIOS compile job and unit test job to run periodically at fixed times, dates, or intervals. For example, the cron script can be called every hour. If during the previous hour an XIOS commit is done, then the newer version of XIOS source code will be updated, afterwards, the compile job will be launched, and then the unit tests will be launched. XIOS developers will receive an email notification at the end of the cron script and can check the web page for newly added information.

We mention here that all the work related to the automation is lodged in the XIOS SVN repository[2]*. The executable program (generic_testcase.exe) is generated by the source code generic_testcase.f90 located at XIOS/trunk/src/test/. The compilation scripts, the unit tests files, and the web page html files are all stored in the folder XIOS/trunk/xios_test_suite.*

3. Difficulties overcome

The automation process as well as the web page report were originally planned to be generated and maintained by Jenkins. However, it was not available on Jean-Zay nor on Irène. That is why we designed our own automation mechanism and the web interface. Though it took us some time for the development, the fact that we are not dependent on a third-party library gives us more freedom.

4. Next steps

The next steps of our work on the XIOS unit tests consists of adapting the user-defined configure file to a more hierarchical form. Take the compile phase configuration file as example, we define:

arch=X64_Irene_arch1, X64_Irene_arch2
mode=prod, debug

With this input, we will have 4 different configurations in total for compiling XIOS. However, one may not be interested in compiling XIOS with X64_Irene_arch2 in debug mode. To do so under the current circumstances, we have to define two different files: a file containing:

arch=X64_Irene_arch1
mode=prod, debug

and another file with:

2 http://forge.ipsl.jussieu.fr/ioserver/browser/XIOS/trunk

arch=X64_Irene_arch2
mode=prod

These two files have to be given to the automation procedure manually and sequentially which destroys the interest of automation. With a hierarchical form, we will have more freedom in defining the configuration file. For example, we can write in the file:

arch=X64_Irene_arch1, mode=prod, debug
arch=X64_Irene_arch2, mode=prod

Thus we will still have only one configuration file and everything can go on automatically.

The same ideas apply to the external file used for configuring the unit tests of XIOS. We may want to continue the test if and only if certain unit tests succeed. For example, we can have two functionalities A and B where B is a derivative of A. If A cannot pass the test, then it will be useless to test B and the test job should stop here. For the moment, all tests are considered to be independent so the status of one test have no influence on the rest of the tests. The tests dependency will be managed with the hierarchical input files.

We will also work on improving the configuration possibilities of the test phase. We will increase the number of configuration variables (which is 7 at the moment for the unit tests part) in order to make XIOS more flexible and adaptive to much more complex situations.

An important issue related to the increasing number of configurations lies in the launch mode of the unit tests job. For the moment, all the unit tests are launched one after the other. However, when we increase the number of configurations, and the number of unit tests themselves, this mode of job launch will no longer be appropriate. We want independent tests be executed in parallel while keeping the dependent tests in sequential mode.

As the development of XIOS goes on, we may introduce new algorithms and functionalities. So one part of our future work is to regularly update the unit tests.