Automated regression testing and code coverage analysis of the CP2K application

# Testing the big supercomputing Hydra

*Marko Mišić*

Developing a large scale, parallel application is a very demanding task, especially if you want your application to run on a wide range of architectures. In order to do so, you need a decent testing environment to bolster your chances of getting everything to work right.

You have probably heard the story of St. George and the Dragon. According to legend, the Dragon caused many problems for the ancient city of Lasia, somewhere in the Middle East. Dragons are mighty, fire-breathing creatures, but even in those auld times, there were brave men like St. George who slayed the Dragon, saved the city, and became a legend.

In modern days we do not believe in those mythical creatures, but rather use them as a metaphor of something really huge and powerful. Nowadays, supercomputers are somehow like dragons – they are mighty and powerful, consuming vast amounts of energy and producing heat, but still vulnerable due to the errors in software. Like the dragon Smaug from J. R. R. Tolkien's 'The Hobbit' who had a weak spot, an 'Achilles' heel' in his armour that eventually led to his defeat.

Supercomputers will not disappear, that is certain, but rather continue to evolve, leading to diversity in High Performance Computing ecosystem, and thus posing problems to application developers. Instead of dealing with one sort of "supercomputing beast", they have to deal with many in order to run their application on different platforms.

Figuratively speaking, it makes more sense to think of a parallel, large scale application as the dragon with more than one head, or, if you like, as more powerful mythical creature – the Lernaean Hydra. As you remember, Hydra had many heads – just like the supercomputers developers use to build, test and execute their parallel applications. It is one of the reasons why testing of large scale applications is becoming increasingly important.

## Instead of introduction

Before going into details of the testing process, it is worth becoming familiar with the application itself. At least, it is good to know something about its scale in order to be aware of the potential caveats.

CP2K is complex scientific application to perform atomistic and molecular simulations of solid state, liquid, molecular, and biological systems. It provides a general framework for different methods including force fields, and *ab initio* models like Density Functional Theory, Hybrid DFT-Hartree-Fock, and post-HF methods. CP2K is freely available under GPL licence on its website [1].

The application itself is written in Fortran 95 with the support for both serial and parallel execution. It can be executed on a wide range of architectures using different parallel programming models like Message Passing Interface (MPI), OpenMP for threading, and a hybrid of the two. Key computational parts of the application are implemented for GPU execution using NVIDIA Compute Device Unified Architecture (CUDA), and there is an ongoing research effort to port the application to new Intel Xeon Phi platform.

If you look even closer at the CP2K code, you will see that it has a really large code base, consisting of more than 900,000 lines of code. The code is supplied with a suite of around 2,400 example input files which can be run as a regression test of the code's functionality, both for the benefit of developers and for users building the application for the first time on a new system. So you see that the application is really heterogeneous, as well as the hardware on which it can be executed. Furthermore, differences exist between compilers for the same language, making the whole thing even more complex.

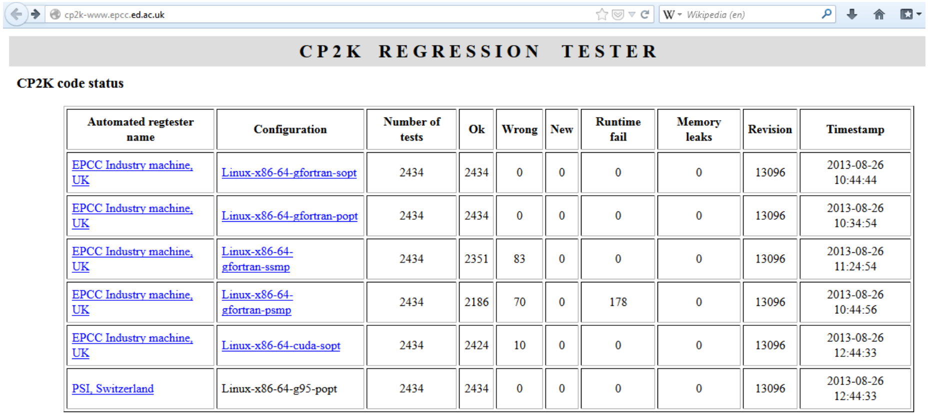
## Regression testing at a glance

Traditionally, regression testing can be described as a verification process after changes like enhancements, patches or configuration changes have been made to the system. Essentially, the role of regression testing is to ensure that no new faults or bugs have been introduced with new versions of the code, and to make sure that changes in one part of the code do not affect other parts. The CP2K community of developers is active daily, making on average two commits to the code base per day in the last 12 months, thus making the need for extensive regression testing of the code even more important.

CP2K regression testing is done with a suite of around 2,400 example input files executed by a Linux bash script solely written for that purpose. The script checks the code for build errors and correctness, reporting any wrong results, runtime failures or memory leaks.

During the process, the regression testing script updates the code to the latest version submitted to SVN repository, and then compiles it. If build process goes fine, the script executes tests one by one, and checks each result obtained against a known good value.

Although nicely written, the whole process did not follow rapid development of the application itself over the years. CVS version control system was replaced with SVN in 2011. CP2K was ported to new platforms. The developers realized that there was no way to execute only failed tests, and generally it missed new options to support growing code base. For example, due to the nature of floating-point calculations in multithreaded environments, you can get results that slightly vary from the reference ones. Those results are not erroneous, so regression testing needed a way to tolerate all those that fail by less than a specified (small) margin.



CP2K REGRESSION TESTER

CP2K code status

| Automated regtester name | Configuration | Number of tests | Ok | Wrong | New | Runtime fail | Memory leaks | Revision | Timestamp |
|---|---|---|---|---|---|---|---|---|---|
| EPCC Industry machine, UK | Linux-x86-64-gfortran-sopt | 2434 | 2434 | 0 | 0 | 0 | 0 | 13096 | 2013-08-26 10:44:44 |
| EPCC Industry machine, UK | Linux-x86-64-gfortran-popt | 2434 | 2434 | 0 | 0 | 0 | 0 | 13096 | 2013-08-26 10:34:54 |
| EPCC Industry machine, UK | Linux-x86-64-gfortran-ssmp | 2434 | 2351 | 83 | 0 | 0 | 0 | 13096 | 2013-08-26 11:24:54 |
| EPCC Industry machine, UK | Linux-x86-64-gfortran-psmp | 2434 | 2186 | 70 | 0 | 178 | 0 | 13096 | 2013-08-26 10:44:56 |
| EPCC Industry machine, UK | Linux-x86-64-cuda-sopt | 2434 | 2424 | 10 | 0 | 0 | 0 | 13096 | 2013-08-26 12:44:33 |
| PSI, Switzerland | Linux-x86-64-g95-popt | 2434 | 2434 | 0 | 0 | 0 | 0 | 13096 | 2013-08-26 12:44:33 |

**Figure 1:** Automated regression tester front page

## Automation is the right way

In software development, especially in agile development methodologies, it is considered good practice to check your code for bugs regularly. Ideally, you would like to check your code after every commit to SVN repository, so that you can catch potential bugs after any change in the code.

Regular, automated regression testing was done for CP2K only at one site in Switzerland. Furthermore, the testing was done only for the MPI implementation with two processes and the g95 compiler which is a bit outdated. This configuration clearly tested only a subset of CP2K's functionality, which we proved to be true by running code coverage analysis. That analysis showed that only about half of the code is included in MPI version, thus leaving some old and new features, like OpenMP and CUDA code, out of reach. Also, the code was not tested with any other compiler, so it posed problems during the porting of the application to Intel Xeon Phi, which uses only Intel compilers, just to give an example.

## Panacea to our ills

In order to improve the testing of CP2K code, we decided to make automated regression testing more comprehensive and robust. We wanted to set it up for a range of architectures and compilers, including different serial and parallel versions - MPI, OpenMP, CUDA, hybrid, etc. There was also a need to include more compilers, so that tester can catch a wider range of build errors.

Also, quality of testing can be measured by actual percentage of code being tested and exercised during the testing process. That is why we wanted to analyze the CP2K code with code coverage tools, to actually see how well the tests cover the code. And in the end, we needed to present all the information gathered through regression testing process in clear and obvious way.

To do all of that, the existing regression test environment needed a major facelift - the first for several years. The legacy automated regression tester needed to be completely rewritten, and to cover all those different platforms we needed a way to do it easily.

## A long road to automated testing

Setting up automated regression testing is not an easy task as it might look at the first sight. Although there are external tools like Jenkins, BuildBot or Hudson that might help you with that task, they are mostly used in complex application software developments that follow extreme programming methodologies. We needed a simpler, yet efficient way, to support execution of CP2K on remote hosts (supercomputers), while processing the results on a local web server and showing them to the world.

To achieve this, we developed a new set of bash scripts, configuration files, and HTML templates. The core of the tester consists of two layers in two separate scripts. The outer layer is responsible for processing and presentation of the results, and it constantly monitors for changes in the code repository. Once the code has changed, it invokes inner layer which takes care of the regression

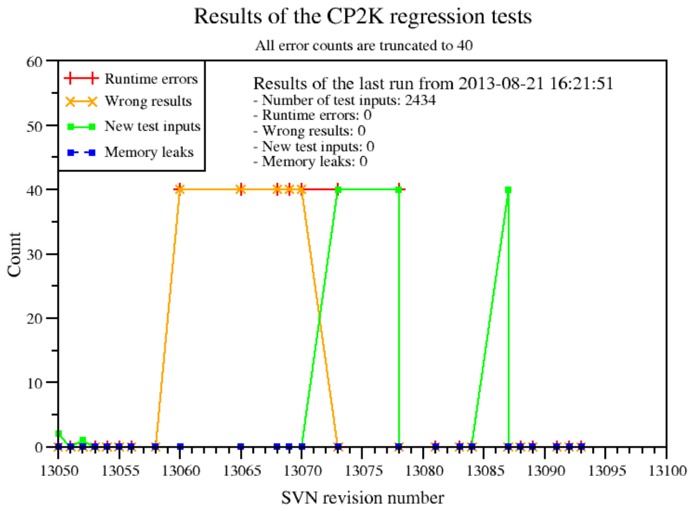


**Figure 2:** A detailed view of particular regression tester

testing using the main regression testing script and a set of configuration files for each particular system. The two layers communicate only at strictly defined points through command line options and files that contain the results.

This architecture supports both local and remote testing, as two layers do not have to execute on the same machine. With minimal, localized configuration changes, it is possible to execute the inner layer on a remote system, while keeping outer layer on a local web server. Supercomputers usually cannot act as web servers, so this was a feasible solution to aggregate and process results in one place, while having the flexibility to test CP2K application on different system.

The whole system is easy to set up and configure, since all changes are localized in configuration files. A helper script is written to aggregate all the results in one page that is shown in Figure 1. There you can see information about the code status for the most recent code revision for each tested configuration. A more detailed view for every particular regression test environment is available in a separate page. There you can find the history of the tester (Figure 2), the last ten regression testing reports and more. Also, there is a section on the website containing information about code coverage which is generated by the very handy LCOV tool.

## Lessons learned

Many problems have been encountered during the work of this project. First of all, it is not easy to improve legacy code while maintaining the backward compatibility. We found that problem while working both with main regression testing script and automated regression tester.

Second, deploying a large scale application to different systems can be really hard. Those applications rely heavily upon third-party libraries, and they are usually sensitive to different compilers or even different versions of the same compiler. This is especially important for those supercomputing systems that are heterogeneous and consist of different backend nodes, as software stack might not be unified across all the nodes. At some point, you might end up with code able to compile and execute on one node, but not the one you need or have access to!

In the end, remote execution of jobs is always tricky as more levels of indirection you have, the more you are prone to errors if you are not familiar with the remote system. Since supercomputers are usually very different “creatures”, you have to cope not only with different operating system installed, but also with different job submission (batch) systems.

## Instead of conclusion

At this point, we can say that our project achieved its most important aim – to provide the developers of CP2K with comprehensive testing and information about their code status. The automated regression tester has been deployed to five new platforms and it aggregates results from six platforms in total, and also provides code coverage data.

Of course, a testing system like one we implemented should constantly be improved to keep the pace with ongoing application development. Future work will surely include a more detailed analysis of code coverage information, and should cover more platforms, with Intel and Cray compilers especially in mind.

In the end, we can say that we tamed our supercomputing Hydra. But, beware that if you do not keep pace with your application, your Hydra may have grown new heads next time you wake up.

### PRACE SoHPCProject Title

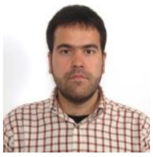
Multi-platform parallel code coverage and regression testing with CP2K

### PRACE SoHPCSite

Edinburgh Parallel Computing Centre, UK

### PRACE SoHPCAuthor

Marko, Mišić,
University of Belgrade,
School of Electrical
Engineering, Serbia



### PRACE SoHPCMentor

Iain Bethune,
Edinburgh Parallel Computing Centre, UK

### PRACE SoHPCContact

Marko, Mišić, University of Belgrade,
School of Electrical Engineering, Serbia
Phone: +381 64 294 86 55
E-mail: marko.misic@etf.bg.ac.rs

### PRACE SoHPCSoftware Applied

CP2K

### PRACE SoHPCMore Information

http://cp2k-www.epcc.ed.ac.uk/

### PRACE SoHPCAcknowledgement

Many thanks to Iain Bethune, Ruyman Reyes, Nick Brown, Fiona Reid, Mark Bull, David Henty, and Irina Nazarova from EPCC who helped me during the realization of this project.

### PRACE SoHPCReferences

[1] CP2K, Open Source Molecular Dynamics, http://www.cp2k.org/

[2] CP2K automated regression tester, http://cp2k-www.epcc.ed.ac.uk/