Non regression testing for the JOREK code

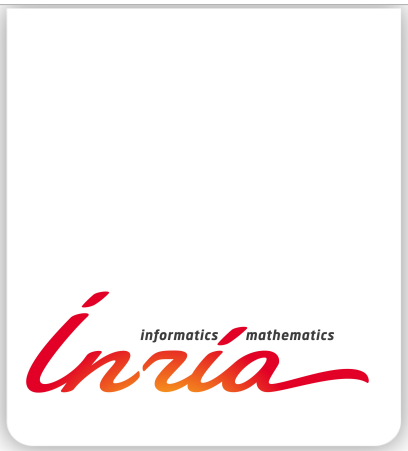
Guillaume Latu, Marina Becoulet, Guilhem Dif-Pradalier, Virginie Grandgirard, Matthias Hoelzl, G. Huysmans, Xavier Lacoste, Eric Nardon, Francois Orain, Chantal Passeron, et al.

Non regression testing for the JOREK code

G. Latu, M. Becoulet, G. Dif-pradalier, V. Grandgirard,
M. Hoelzl, G. Huysmans, X. Lacoste, E. Nardon, F. Orain,
C. Passeron, P. Ramet, A. Ratnani

RESEARCH
REPORT
N° 8134
November 2012
Project-Team HIEPACS

ISSN 0249-6399 ISRN INRIA/RR--8134--FR+ENG



Non regression testing for the JOREK code

G. Latu*, M. Becoulet*, G. Dif-pradalier*, V. Grandgirard*,
M. Hoelzl†, G. Huysmans‡, X. Lacoste§, E. Nardon*, F. Orain*,
C. Passeron*, P. Ramet§, A. Ratnani*

Project-Team HIEPACS

Research Report n° 8134 — November 2012 — 14 pages

Abstract: Non Regression Testing (NRT) aims to check if software modifications result in undesired behaviour. Suppose the behaviour of the application previously known, this kind of test makes it possible to identify an eventual regression, a bug. Improving and tuning a parallel code can be a time-consuming and difficult task, especially whenever people from different scientific fields interact closely. The JOREK code aims at investing Magnetohydrodynamic (MHD) instabilities in a Tokamak plasma. This paper describes the NRT procedure that has been tuned for this simulation code. Automation of the NRT is one keypoint to keeping the code healthy in a source code repository.

Key-words: nonlinear MHD, Tokamak plasma, Non Regression Testing

* CEA Cadarache, IRFM bat. 513, F-13108 Saint-Paul-lez-Durance
† Max-Planck-Institute for Plasma physics, Boltzmannstr. 2, D-85748 Garching
‡ ITER Organisation, Route de Vinon sur Verdon, F-13115 St-Paul-lez-Durance
§ INRIA, 351 cours de la Liberation - F-33405 Talence

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

351, Cours de la Libération
Bâtiment A 29
33405 Talence Cedex

Tests de non-regression pour le code JOREK

Résumé : Les tests de non regression (l'acronyme anglais est NRT) ont pour objet de vérifier si les modifications apportées à un logiciel conduisent, ou non, à des comportements corrects ou incorrects. Ayant caractérisé et référencé des comportements corrects liés à des scénarii d'exécution précis, ces tests permettent d'identifier durant le processus de développement une éventuelle régression, un bug. L'amélioration et l'optimisation d'un code parallèle est une tâche consommatrice de temps et parfois difficile. Cela est d'autant plus vrai lorsque différents acteurs interagissent étroitement, dans notre cas : des physiciens, des mathématiciens, des informaticiens. Le code JOREK traite d'instabilités liées à la Magnétohydrodynamique (MHD) dans des plamas de Tokamak. Ce papier décrit la procédure de NRT qui a été mise en place dans ce code de simulation. L'automatisation des NRT est un point essentiel pour conserver, dans la durée, un code sain dans un dépôt de sources.

Mots-clés : MHD non-linéaire, plasma de Tokamak, tests de non-regression

1 Introduction

This paper deals with a Non Regression Testing (NRT) tool for a code dedicated to the simulation of MHD instabilities relevant to magnetically confined fusion.

Magnetohydrodynamic (MHD) stability and the avoidance of plasma disruption - rapid loss of plasma energy and abrupt termination of the plasma current caused by global MHD instability growth - are key considerations to the attainment of burning plasma conditions in ITER (a large fusion reactor `www.iter.org`). Realization of adequate MHD stability and careful control of the plasma operation will be critical ITER operation issues: MHD instabilities can damage components of tokamak walls.

Numerical simulations play an important role in the investigation of the non-linear behaviour of these instabilities and can help for interpretating experimental observations. In the framework of non-linear MHD codes, targeting realistic simulation requires: to model complex geometry, to handle a large gap between the different time scales relevant to plasmas, to address full 3D simulation. Computational time needed to run 3D MHD code named JOREK necessitates parallel computing in order to get reduced restitution time for the user [CH08, HMH+12].

We describe here procedures and solutions to overcome a lack of NRT and benchmarking tools for JOREK users. The aims of such improvements are twofold: first, to keep the main JOREK trunk healthy in the version control system, second, to have a way to compare execution times and results of runs launched on different supercomputers.

In this document, we will often refer to the SVN repository where JOREK code is stored. This repository contains also all the materials and scripts that are described herein:

`scm.gforge.inria.fr/svnroot/aster`

This work was made possible by a ANR grant and the accesses to several parallel machines. The authors acknowledge the support of the French Agence Nationale de la Recherche (ANR) under reference ANR-11-MONU-0002 (ANEMOS project). Computations have been performed at the Mésocentre d'Aix-Marseille Université (France), on PLAFRIM Bordeaux (France), on IFERC Rokasho (Japan).

2 NRT

To begin with, we will sketch up the JOREK environment in order to explain the choices made for the Non Regression Testing. Then, the method used to achieve NRT is explained.

2.1 Numerical components

Spatial discretization JOREK is a MHD three dimensional fluid code that takes into account realistic tokamak geometry. High spatial resolution in the poloidal plane is needed to resolve the MHD instabilities at high Reynolds and/or Lundquist numbers. Bezier patches (2D cubic Bezier finite elements) are used to dicretize variables in this plane. Hence, several physical variables and their derivatives have a continuous representation over a poloidal cross-section. The periodic toroidal direction is treated via a sine/cosine expansion.

Set of equations Some of the variables modelled in JOREK code are: the poloidal flux Ψ, the electric potential u, the toroidal current density j, the toroidal vorticity ω, the density ρ, the temperature T, and the velocity $v_{parallel}$ along magnetic field lines. Depending on the model choosen (hereafter denoted **MODEL** which is a simulation parameter), the number of variables and the number of equations on them are setup. At every time-step, this set of reduced MHD equations is solved in weak form as a large sparse implicit system. The fully implicit method leads to very large sparse matrices. There are some benefits to this approach: there is no *a priori* limit on the time step, the numerical core adapts easily on the physics modelled (compared to semi-implicit methods that rely on additional hypothesis). There are also some disadvantages: high computational costs and high memory consumption for the parallel direct sparse solver (PASTIX or others).

Time integration scheme The temporal discretization is performed by a fully implicit second-order linearized Crank-Nicholson scheme. Two operators C and D are matrices that

describe the set of MHD equations in weak form:

$$\frac{\partial C(\vec{u})}{\partial t} = D(\vec{u}), \quad \text{discretized by } \left(\frac{\partial C(\vec{u_n})}{\partial u} - \frac{1}{2}\delta_t \frac{\partial D(\vec{u_n})}{\partial u}\right) \delta\vec{u} = D(\vec{u_n})\,\delta t \tag{1}$$

$$\text{Let us denote } A = \frac{\partial C(\vec{u_n})}{\partial u} - \frac{1}{2}\delta_t \frac{\partial D(\vec{u_n})}{\partial u}, \quad b = D(\vec{u_n})\,\delta t, \quad \text{then} \tag{2}$$

$$A\,\delta\vec{u} = b \tag{3}$$

Where $\vec{u_n}$ is the set of variables at time step n, and $\delta\vec{u} = \vec{u_{n+1}} - \vec{u_n}$ is the unknown. In the following, we will refer to A as the sparse matrix that should be solved, and b the right hand side of the problem.

Equilibrium Each JOREK simulation begins with the solving of the static magnetic equilibrium equation (so-called Grad-Shafranov equation) in the 2 dimensions of the cross-section plane. A keypoint is the ability to handle magnetic equilibria which include an X-point. High accuracy is needed to have a correct representation of this equilibrium and avoid spurious instabilities whenever the whole simulation 3D+t is launched.

JOREK is able to build a Bezier finite element grid aligned with the flux surfaces both inside and outside the separatrix (*i.e.* the flux surface containing the X-point). This strategy allows one to improve the accuracy of the equilibrium representation. The flux surfaces are represented by sets of 1D Bezier curves determined from the numerical solution of the equilibrium. The Grad-Shafranov solver is based on a Picard's iteration scheme.

After the Grad-Shafranov solving step, a supplementary phase is required: the time-evolution equations are solved only for the $n=0$ mode (the first toroidal harmonic, *i.e.* purely axisymmetric) over a short duration. First, very small time-steps are taken, then they are gradually increased. This process allows the plasma equilibrium flows to establish safely in simulations involving a X-point.

Sparse solver & preconditionning A direct parallel sparse matrix solver (PASTIX or others) is used to solve the large linear systems inside JOREK. In order to minimise the memory requirements of the fully implicit solver and to access larger domain sizes, a preconditioner accompanied with a GMRES iterative solver have been included a few years ago. Preconditioning transforms the original linear system $A\,x = b$ into an equivalent one which is easier to solve by an iterative technique. A good preconditioner P is an approximation for A which can be efficiently inverted, chosen in a way that using $P^{-1}\,A$ or $A\,P^{-1}$ instead of A leads to a better convergence behaviour. Usually, GMRES iterative solver is applied in collaboration with a preconditioner. The preconditioner typically incorporates information about the specific problem under consideration.

The JOREK *physics-based* preconditioner has been constructed by using the diagonal block for each of the `n_tor` Fourier modes in the toroidal direction of the matrix A presented earlier in Eq. 3. The preconditioner represents the linear part of each harmonic but neglects the interaction between harmonics (similar to a block-Jacobi preconditioning on a reordered matrix). So, we set many coefficients of the original matrix A to zero, in order to get a block-diagonal matrix with m independent submatrices on the diagonal. The preconditioner P consists in the composition of m independent linear systems $(P_i^\star)_{i\in[1,m]}$, with $m = \frac{n\_tor+1}{2}$. Practically, the set of processors are split in m independent MPI communicators, each of them treats only one single linear system $P_i^\star$ with a sparse direct solver. This preconditioned parallel approach avoids large costs in terms of memory consumption compared to the first approach that considers the whole linear system to solve (it saves the memory needed by the sparse solver to store the decomposition of big matrix A - *e.g.* L, U factors). Nevertheless, the whole linear system A has to be built (in parallel) in order to perform the matrix-vector multiplication needed by the GMRES. But, the cost in memory and in computation is far less than invoking the parallel sparse solver on the large linear system A.

This strategy improves the scalability (m inpendent systems), and the parallelisation performance of the code. The bad point is that in some specific circumstances, the iterative scheme may not converge.

2.2 Parameters

From the user perspective, one can distinguish three sets of parameters in the JOREK code:

- **Category A - Hardcoded.** A first set of parameters is fixed inside the source code. For example: the selected physics model (**MODEL** parameter), the number of harmonics which defines the discretization along toroidal dimension (`n_tor`), and some others (`n_period`, `n_plane`, `n_vertex_max`, `n_nodes_max`, `n_elements_max`, `n_boundary_max`, `n_pieces_max`, `gmres_max_iter`, `iter_precon`, ...).

- **Category B - Input file.** A second set of parameters are given through the standard input to JOREK at launch time. Some examples of such input files can be found in `jorek2/trunk/namelist` directory in the JOREK repository.

- **Category C - Environment.** A third set of parameters consists in the parallel environment at launch time. For example, the following data constrain the execution time and impact a little bit the numerical results: the number of MPI processes, the number of shared memory nodes, the number of OpenMP threads, the set of libraries used (MPI, sparse solver library).

Several kinds of simulations can be undertaken with this tool. We will mainly address in this document few simulations that are parametrized by the following inputs (mainly of category B - input file):

- **Geometry** Three main configurations are available concerning the geometry in the poloidal plane: circular cross-section (parameter `xpoint=.f.`), single X-point (parameter `xpoint=.t.`), two X-points (not detailed here). Other fine parameters permit to fit the geometry to a realistic configuration.

- **Spatial domain size** Some parameters set up the computational domain. The discretization highly impacts the memory and computational costs. Poloidal domain is sized by `n_flux` the number of flux surfaces (radial coordinate), `n_tht` the number of points in the angular direction (equivalent to θ). In the toroidal direction, `n_tor-1` sizes the number of sine/cosine components that will be computed. The `n_tor` parameter is in the category A - hardcoded. JOREK code is able to handle the increase of `n_tor` parameter between a checkpoint and a restart.

- **Time axis** Depending on the simulation kind and the evolution of the simulation, the time step duration can be adjusted. The timestep `tstep` is typically fixed from a fraction of an Alfven time to thousands of Alfven times. The number of time steps in a single run is set by `nstep`. There is another way to specify both number of time steps and durations through the parameters `tstep_n` and `nstep_n` which are vectors. It helps the user in order to define complex scenarii with multiple timesteps inside a single run.

2.3 Scenarii definition

Edge localized modes (ELMs) are intermittent relaxations of the edge transport barrier. During these events, particles and energy are ejected from the plasma edge into the surrounding low density envelope, the SOL (scrape-off layer). The associated transient power loads create problems during the operation of tokamak reactors. MHD codes, such as JOREK, can describe the ELM evolution.

One of the relevant instabilities for the ELMs are the MHD peeling-ballooning modes, but other kinds of instabilities or physics phenomena can be studied with JOREK [HC07, NBHC07] (kink modes, tearing modes ...). A typical simulation begins from an axisymmetric equilibrium ($n=0$) on top of which a small $n \neq 0$ perturbation is initialised. First, an initial linear phase of exponential growth shows a rapid energy increase (when a mode is unstable), starting from a very low level of energy (a little bit noisy). Kinetic and magnetic energies can have quite large growth rates during the linear phase. For some identified cases, the theoretical growth rates that should be obtained are known. After a while, some Fourier modes saturate non-linearly (growth rates are droping).

In order to track a simulation and analyse its time evolution, we will look at the dynamics of kinetic and magnetic energies. Somehow, they constitute a valid signature of a given simulation scenario. The physics phenomena strongly constrain these energies. Other criteria could also be taken into account in the future, in order to track fine differences between two runs that reproduce the same scenario (for example spatially localized phenomena).

A typical scenario execution is decomposed in the following way. First, the equilibrium is computed separately within a single JOREK execution. Second, a set of `tstep_n` and `nstep_n` vectors defines a *subsequence* of time steps that will be used in a second JOREK execution. A

complete scenario is composed from one *subsequence* to several such *subsequences*. Let us notice that between two successive JOREK executions, the parameter `n_tor` can be modified in order to increase/decrease toroidal resolution. A set of scenarii definitions have been encapsulated in `jorek2/trunk/util/launch_tests.sh`. For each scenario, the following data are fixed: the *input* parameter file, the set of vectors `tstep_n` and `nstep_n`, the times where `n_tor` is changed, and the series of JOREK executions (2 to 9 successive runs).

2.4 Metric

In order to build *Non Regression Testing* for a whole simulation code[1], one mainly needs two basic tools: 1) a methodology to replay a given scenario for the code, 2) a way to measure the differences between two runs of the same scenario with respect to test selection criteria. The second point involves the definition of a *Metric* that captures numerically the distance beween two scenario execution traces. Ideally, the *Metric* should also allow users to identify problems, help them in the process of improving the code, and provide a way to detect bugs quickly.

Reproducibility is the degree of agreement between measurements conducted on replicate scenario. While repeatability of scientific experiments is desirable, this is not always an easy task, even in the field of numerical simulations. Typically in the JOREK code, there are at least four reasons that limit someone to reproduce the same results upon demand:

- OpenMP introduces a dependance on threads scheduling inside processes that may alter a bit numerical results from one run to another,
- Global summation with MPI of distributed arrays is most susceptible to nonreproducible rounding errors (*e.g.* addition in MPI_Reduce is not strictly commutative),
- The iterative solver used for the large linear systems depends on a threshold to stop the iteration loop, this nonlinearity is able to amplify numerical noise,
- In many simulations, the starting point of the linear phase is dominated by numerical noise because system state is not far from a perfect equilibrium; then, the very beginning of a simulation run is expected to be very difficult to reproduce. Nevertheless, the rest of the simulation is dominated by a strong signal with much less numerical noise and this second part is reproducible.

So, given that the application is nondeterministic, the difficulty is to discriminate differences between two runs that are fully acceptable (coming from nonreproducibility), from differences that are likely to originate from a bug.

The metric we have chosen is the following: *a time series of the growth rate of magnetic and kinetic Fourier modes on a specific time interval.* The number of Fourier modes and the time interval are test-case dependent. The time interval will be fixed such that the very beginning of the simulation is excluded. We will also not keep the long term evolution of the non-linear saturation because small noise can be amplified and therefore comparison is notably biaised. Finally, only a time interval in the linear phase is kept for this metric. Practically, this means that we have for each reference scenario: a time interval $[t_{start}, t_{end}]$ and a threshold thr, a time series S of the growth rates of M Fourier modes. Let us denote $s_A^{t,m}$ the growth rates of Fourier mode $m \in [1, M]$ at time step $t \in [t_{start}, t_{end}]$ of simulation S_A. To compare two different runs A and B reproducing the same scenario, the metric consists in comparing S_A and S_B with the following criteria:

$$\forall t,\ \forall m, \qquad \frac{2\,|s_A^{t,m} - s_B^{t,m}|}{|s_A^{t,m}| + |s_B^{t,m}|} < thr$$

The difference between two growth rates coming from two runs A and B should be below a given threshold percentage fixed by thr. If it is not the case, we will consider that A and B are different, *i.e.* one run has a wrong behaviour.

[1] Non Regression Testing involve an entire code whereas unitary tests involve subroutines or subparts of the code.

2.5 Parallel job launch

For the sake of simplicity, a complete scenario is executed in only one single parallel job. If it was not the case, one would have to chain multiple jobs which requires some specific parameters to give to the batch scheduler of the parallel machine. In order to automate the NRT as much as possible, the compilation and execution are done within the same parallel job (three models are available: 199, 302, 303). There are two possibilities to launch a JOREK reference scenario:

a) Launch a parallel job over 4 SMP nodes (one MPI process per node) and inside this job compile all executables and then run the scenario with the following commands (it requires that you have configured your Makefile.inc already):

```
$ jorek2/trunk/util/compile_test.sh <MODEL> # compile JOREK for some <MODEL>
$ export PRERUN='export OMP_NUM_THREADS=8'  # command to launch before mpirun
$ export MPIRUN='mpirun -np 4'              # mpirun command
$ export BASEDIR='/scratchdir/username'     # root where the new dir will be created
$ jorek2/trunk/util/launch_test.sh <DIRECTORY PREFIX><MODEL> # run the scenario
```

Let us remark that on some machines (*e.g.* Helios-IFERC machine), the compilation process (*i.e* `compile_test.sh`) can not be done on compute nodes. Therefore the compilation should be done on login nodes, before the job launch.

b) If `subjorek` is configured for the target parallel machine, you can alternatively use the following command:

```
jorek2/trunk/run_jorek/nrt_run.sh all <DIRECTORY PREFIX><MODEL>
```

The reference scenario will be executed and will generate a directory named `<DIRECTORY PREFIX><MODEL>` that contains all outputs of a standard JOREK run. In this directory, the main data that we will used to compare numerical results to another run is stored in the file `macroscopic_vars.dat`.

Some examples of batch scripts to run a reference scenario are given in `benchmark/batch` directory.

2.6 Comparing against reference scenario

To compare a run against a reference scenario, you have to download some files from the SVN repository with a command such as

`svn co svn+ssh://<LOGIN>@scm.gforge.inria.fr/svnroot/aster/benchmark`

A set of reference scenarii is located in `benchmark/model*` directories. The data used to compare with the metric a JOREK run against a reference scenario is encapsulated into the script file `jorek2/trunk/util/nrt_compare.sh`. This script requires the access to a tiny tool named `numdiff` that is easy to compile and install on a LINUX/UNIX system (`http://ftp.igh.cnrs.fr/pub/nongnu/numdiff/`).

For example, let suppose that you have reproduced the reference scenario of model `199` with the command:

```
$ jorek2/trunk/util/launch_test.sh new199
```

After the job completion, the directory `new199` contains the results of the simulation you have launched. To compare your data with the reference case stored into `aster/benchmark/model199/helios_a`, you just have to do this:

```
$ nrt_compare.sh 199 new199 aster/benchmark/model199/helios_a
OK (nb lines compared: 53/53)

# gnuplot commands to look at growth rates that have been compared :
set key autotitle columnhead;
set auto; plot 'f1' u 1:2 ls 1, 'f2' u 1:2 ls 4; pause -1
set auto; plot 'f1' u 1:3 ls 3, 'f2' u 1:3 ls 6
```

The comparison is successful and 53 time steps over 53 have been compared and verified[2]. To look at the growth rates of kinetic and magnetic energies that have been compared, you can

[2]It is also possible to perform a `nrt_compare.sh` during a simulation and the check will be realized on available data in the `macroscopic_vars.dat` file.

use the gnuplot commands that are mentionned. The data inside `f1` and `f2` files are extracted from growth rates of the file `macroscopic_vars.dat` that is generated by JOREK.

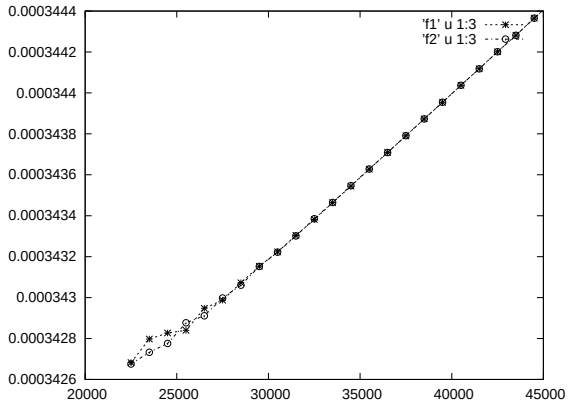


Figure 1: Example of gnuplot output generated by plotting f1 and f2 files with time evolution (abscissa) of growth rates (ordinate)

On the Fig. 1, the `f1` curve corresponds to the growth rate of one Fourier mode (magnetic energy $n = 3$) for simulation `model199/helios_a`, while `f2` corresponds to the same mode on another machine `new199` (= `model199/rheticus_a`). One can see that even the beginning differs slightly (numerical noise is perceptible), the two curves coincide almost exactly above 30 000 Alfven times. The `nrt_compare.sh` script checks between the two simulations if growth rates do not differ much more than 1 percent and says `OK` in this case.

A good methology using NRT is the following: whenever the user wants to perform a `svn commit`, he or she should launch all reference scenarii. If the modification is not supposed to alter the results, `nrt_compare.sh` will say `OK` for each scenario. The database for Non Regression Testing is located in `benchmark/model*`, the user can equally refer to all cases stored into `benchmark/model*` directories as third parameter of the `nrt_compare.sh` call. Once the user has obtained a `OK` for all scenarii, he or she can commit the changes in the SVN repository.

3 Benchmarks

The need for NRT has permitted to define a set of scenarii which are representative of some classical user cases of JOREK (yet, not exhaustive but the list is expected to be extended by users). These scenarii can also be used to benchmark machines and to compare execution times in different parallel configurations (number of MPI processes, threads), on several parallel machines. This information helps the user to determine if the performance on a new machine is far away from other machines or not, and to evaluate the parallel performance.

3.1 Timer measurements

A script `timing_bench.sh` is located in the `benchmark` directory. It facilitates the extraction of timers stored in JOREK log files. For example to look at the elapsed time taken by each JOREK temporal iteration on the two machines `fourmi` and `rheticus` in the `out_loop1` log file, one can execute the command:

```
$ cd benchmark
$ ./timing_bench.sh ITER out_loop1 grep3 model199/fourmi_a model199/rheticus_a
== model199/fourmi_a ( openmpi )
0 # Elapsed time ITERATION : 22.4025180
0 # Elapsed time ITERATION : 10.5183380
0 # Elapsed time ITERATION : 11.3435830
== model199/rheticus_a ( mvapich2 )
0 # Elapsed time ITERATION : 15.2480000
0 # Elapsed time ITERATION : 7.2230000
0 # Elapsed time ITERATION : 7.5020000
```

Several parameters has to be given to this script: 1) the keyword that will be used to perform the grep in the log file (*e.g.* `ITER` to get the time taken by one temporal iteration), 2) the name

of the JOREK log file (*e.g.* `out_loop1`), 3) optionally `grep` followed by a number specifying the number of lines to keep (*e.g.* `grep3` to keep the 3 first lines), 4) a list of directories where the script will lookup the log files to process (*e.g.* `model199/fourmi_a model199/rheticus_a` to look into theses two directories).
The outputted numbers show that the `rheticus` machine performs a single JOREK iteration a little bit faster than the `fourmi` machine. The two machines that have been used for the runs are described in `README.txt` files.

```
$ cat model199/fourmi_a/README.txt
Machine : PLAFRIM/fourmi
Location : Bordeaux - France
Processor : Intel(R) Xeon(R) CPU X5550 @ 2.67GHz
Cores/node : 8
Author : Latu
Comment : openmpi
Jorek fortran compiler : ifort 11.1
Jorek SVN revision : 666
Jorek specific option : -
Pastix compiler : icc 11.1
Pastix SVN revision : 3564
Pastix specific option : -
MPI library : openmpi/1.4.4
BLAS library : GotoBLAS2
Network : Infiniband QDR : 40Gb/s
Other info : -
```

```
$ cat model199/rheticus_a/README.txt
Machine : Rheticus
Location : Marseille - France
Processor : Intel(R) Xeon(R) CPU X5675 @ 3.07GHz
Cores/node : 12
Author : Latu
Comment : mvapich2
Jorek fortran compiler : gfortran 4.4.6
Jorek SVN revision : 666
Jorek specific option : -
Pastix compiler : icc 12.1.0
Pastix SVN revision : 3564
Pastix specific option : -
MPI library : mvapich2-1.7
BLAS library : atlas
Network : Infiniband
Other info : -
```

At present day, a reduced set of timers are accessible. Here is a list of these timers:

- **construct_matrix**: time elapsed to compute all matrix coefficients of the global system A and to build the right hand side vector b (see Eq. 3). A domain decomposition is used to parallelize this step, based on distribution of the Bezier finite elements among MPI processes. A communication step is also performed to form the RHS b.
- **distribute**: time to distribute the coefficients of the preconditioner matrix $P_i^\star$ (copied from distributed A) to all master processes, but also to send the local RHS associated to each local $P_i^\star$. There is one master process per harmonic (*i.e.* $\frac{n\_tor+1}{2}$) that receives $P_i^\star$ and its local RHS.
- **coicsr**: time to convert the preconditioner matrix $P_i^\star$ on each master from $(i, j, value)$ format to CSR (standard compressed sparse format). This transform is generally needed to give $P_i^\star$ as input to the sparse parallel solver (actually Pastix or MUMPS or WSMP).
- **analysis**: first phase of the sparse solver to solve linear systems for matrices $P_i^\star$. It comprises an ordering step that reorders the rows and columns, an analysis step or symbolic factorization that determines the nonzero structures of matrix and creates internally suitable data structures.
- **facto**: second phase of the sparse solver, the factorisation computes the L and U matrices (or another decomposition).
- **first solve**: time to solve the first set of linear systems in a single JOREK temporal step (matrices $P_i^\star$), that includes potentially the analysis, the factorization and the solve step (forward and back substitution algorithms). But often, analysis or factorisation can be skipped.
- **gmres/solve**: time needed by GMRES iterative process that solves many times the linear system until a convergence is achieved or a maximum iteration count is reached. Because analysis and factorization are not done again and again, this gmres/solve step is mainly m successive solve steps using $P_i^\star$ matrix.
- **ITER**: time for one temporal iteration of JOREK. It can be viewed as a partial sum of a subset of the previous timing counters.

Depending on the configuration, the compilation and the deployment of JOREK on a particular parallel machine, the relative weight of these timers over the total time can be quite different. For example, some parts scale along with the number of cores used (*e.g.* `construct_matrix`), others do not scale well (*e.g.* `coicsr`).

If you want to store the data obtained from a reference run in the SVN repository (your run is saved into ~/MYDIR_302 for example), you can use the following procedure:

```
$ cd benchmark
$ ./cp_bench.sh ~/MYDIR_302 model302/MYMACHINE_a
```

This will create a directory at the `benchmark/model302/MYMACHINE_a` location and also copy from your directory `~/MYDIR_302` only a reduced set of files that should be conserved (the `macroscopic_vars.dat` file, the standard ouput files of JOREK runs). There will be a `README.txt` file created in the new directory that you can fill in order to give some information on the machine you used.

3.2 Comparing timers

A set of benchmarks of JOREK on several machines and with different settings are presented in this section. It can give some clues to the JOREK user in order to choose a convenient set of parameters during the configuration/compilation of the code on a new machine.

3.2.1 Impact of the MPI library

On Helios machine (IFERC, Rokasho-Japan), available versions of MPI can not use MPI_THREAD_MULTIPLE safely (up to now - september 2012). Then the *funneled* version of PASTIX has to be used (MPI_THREAD_FUNNELED mode of MPI).
Two MPI libraries have been tested in order to determine which one should be preferred: `intelmpi` (version 4.0.3) or `bullxmpi` (version 1.1.14.3).

```
$ ./timing_bench.sh gmres/solve out_loop5 grep2 model302/helios_?
== model302/helios_a ( bullxmpi + FUNNELED )
0 # Elapsed time gmres/solve : 3.8031330
0 # Elapsed time gmres/solve : 3.4623600
== model302/helios_b ( intelmpi + FUNNELED )
0 # Elapsed time gmres/solve : 3.4759830
0 # Elapsed time gmres/solve : 3.3056950
$ ./timing_bench.sh facto out_loop5 grep1 model302/helios_?
== model302/helios_a ( bullxmpi + FUNNELED )
0 ## Elapsed time, facto : 36.4806480
== model302/helios_b ( intelmpi + FUNNELED )
0 ## Elapsed time, facto : 143.5415650
$ ./timing_bench.sh ITER out_loop5 grep5 model302/helios_?
== model302/helios_a ( bullxmpi + FUNNELED )
0 # Elapsed time ITERATION : 151.7923330
0 # Elapsed time ITERATION : 62.5559670
0 # Elapsed time ITERATION : 9.5537380
0 # Elapsed time ITERATION : 10.3767240
0 # Elapsed time ITERATION : 9.5514340
== model302/helios_b ( intelmpi + FUNNELED )
0 # Elapsed time ITERATION : 260.5344900
0 # Elapsed time ITERATION : 167.9943480
0 # Elapsed time ITERATION : 9.0590700
0 # Elapsed time ITERATION : 9.5267600
0 # Elapsed time ITERATION : 9.0850900
```

From the first two commands, one can draw two conclusions:

a) The solve step (with the *funneled* version of Pastix) takes roughly the same amount of time with `bullxmpi` or `intelmpi` MPI libraries.
b) The factorisation is much quicker with `bullxmpi` implementation.

The last command gives the time spent for some temporal iterations. It is clear that `bullxmpi` fastens JOREK a lot compared to `intelmpi`. The first two iterations are dominated by the factorisation step and the `intelmpi` is much slower on this step.

3.2.2 Impact of the Fortran compiler

JOREK numerical core relies partly on PASTIX and BLAS libraries. These libraries should be configured and compiled properly. But, the compilation of JOREK impacts also the performance. We want to evaluate the impact of fortran compiler on timers.

```
$ benchmark/timing_bench.sh ITER out_loop2 grep5 rheticus_gf_302 if_302
== rheticus_gf_302 ( gfortran 4.4.6 )
0 # Elapsed time ITERATION : 157.0370000
0 # Elapsed time ITERATION : 78.8670000
0 # Elapsed time ITERATION : 14.6350000
0 # Elapsed time ITERATION : 14.9580000
0 # Elapsed time ITERATION : 15.1500000
== rheticus_if_302 ( ifort 12.1 )
0 # Elapsed time ITERATION : 150.9672940
0 # Elapsed time ITERATION : 74.2307940
0 # Elapsed time ITERATION : 8.2149000
0 # Elapsed time ITERATION : 9.0045110
0 # Elapsed time ITERATION : 9.8495460
```

Each iteration takes a bit more time with `gfortran` compared to `ifort`. After some investigations in the output files, the responsible for this difference can be found:

```
$ benchmark/timing_bench.sh construct_ out_loop2 grep5 rheticus_gf_302
rheticus_if_302
== rheticus_gf_302 ( gfortran 4.4.6 )
0 # Elapsed time construct_matrix : 11.9530000
0 # Elapsed time construct_matrix : 11.5190000
0 # Elapsed time construct_matrix : 11.4940000
0 # Elapsed time construct_matrix : 11.4600000
0 # Elapsed time construct_matrix : 11.5520000
== rheticus_if_302 ( ifort 12.1 )
0 # Elapsed time construct_matrix : 5.2446580
0 # Elapsed time construct_matrix : 4.8440020
0 # Elapsed time construct_matrix : 4.8604400
0 # Elapsed time construct_matrix : 4.8635080
0 # Elapsed time construct_matrix : 5.4503650
```

The matrix construct subroutine takes more time with `gfortran`. It explains the gap between elapsed time for a complete ITERATION seen previously.

3.2.3 PASTIX thread-funneled versus thread-multiple

The PASTIX sparse parallel solver uses by default the MPI_THREAD_MULTIPLE if the MPI library supports it. The design of this parallel library is based on a fully multi-threaded environment, and the MPI_THREAD_MULTIPLE mode of MPI helps for that. But if the MPI implementation does not handle MPI_THREAD_MULTIPLE, a funneled version of PASTIX is also accessible. This version, available through -DPASTIX_FUNNELED during PASTIX configuration requires only the MPI_THREAD_FUNNELED capability of the MPI library. In this mode, the MPI process may be multi-threaded, but *only the main thread* will make MPI calls.

```
$ cd benchmark
$ ./timing_bench.sh facto out_loop3 grep2 model302/fourmi_? model302/rheticus_?
== model302/fourmi_a ( openmpi )
0 ## Elapsed time, facto : 64.7742300
== model302/fourmi_b ( openmpi + FUNNELED )
0 ## Elapsed time, facto : 50.9117520
== model302/rheticus_a ( mvapich2 )
0 ## Elapsed time, facto : 39.4410000
== model302/rheticus_b ( mvapich2 + FUNNELED )
0 ## Elapsed time, facto : 35.4110000
```

This example show the time taken by the factorisation step performed by PASTIX solver on two machines for model `302`. On the two machines, with two different MPI libraries, the *funneled* version of Pastix is quicker than the *multi-threaded* one.

```
$ cd benchmark
$ ./timing_bench.sh gmres/solve out_loop3 grep1 model302/fourmi_? model302/rheticus_?
== model302/fourmi_a ( openmpi )
0 # Elapsed time gmres/solve : 1.5070440
== model302/fourmi_b ( openmpi + FUNNELED )
0 # Elapsed time gmres/solve : 4.4459140
== model302/rheticus_a ( mvapich2 )
0 # Elapsed time gmres/solve : 1.3680000
== model302/rheticus_b ( mvapich2 + FUNNELED )
0 # Elapsed time gmres/solve : 3.7280000
```

Concerning the gmres/solve iterative process that mainly involves the solve step of the Pastix solver, the *funneled* version is nearly three times slower than the multithreaded version. The performance ratio is the opposite of the one made for the factorisation.

As a conclusion, if you have access to a library that has both `MPI_THREAD_FUNNELED` and `MPI_THREAD_MULTIPLE` available, you have to check which configuration gives you best performance for your runs. If your JOREK jobs are mainly dominated by `solve` performance `MPI_THREAD_MULTIPLE` should be better, but if your runs are dominated by `facto` performance `MPI_THREAD_FUNNELED` should be faster.

3.2.4 Parallel strong scaling

The `timing_bench.sh` tool can be used for parallel benchmarking. In order to evaluate parallel performance of a code, a classical test is the strong scaling. Given a fixed test case (we have taken the reference scenario of model 302 with X-point geometry and `n_tor=3`), the number of cores is doubled successively. Ideally, the timing of parallel subroutines should be successively divided by a factor two. For this experiment, we have set the JOREK compilation flag `-DUSE_BLOCK` that shortens the analysis step of the parallel sparse solver. The `rheticus` machine of Mésocentre de Marseilles was used.

Nb cores	12	24	48	96	192
Nb nodes	1	2	4	8	16
Nb MPI proc.	4	4	4	8	16
Nb threads	3	6	12	12	12
Steps					
`construct_m`	*14.74*	7.53	4.13	2.08	1.47
`coicsr`	*0.41*	0.30	0.29	0.31	0.31
`distribute`	*3.30*	2.90	2.93	2.50	2.25
`analysis`	*2.03*	1.69	1.71	2.16	2.40
`facto`	*265.8*	67.2	41.5	22.7	14.5
`gmres/solve`	*18.5*	2.53	1.76	1.33	0.73
`ITER`	*308.7*	83.4	53.3	31.9	22.5

Table 1: Timing in seconds for the first iteration (reference scenario model 302, `out_loop3` file)

Nb cores	12	24	48	96	192
Nb nodes	1	2	4	8	16
Nb MPI proc.	4	4	4	8	16
Nb threads	3	6	12	12	12
Steps					
`construct_m`	15.7	7.20	3.90	2.07	1.20
`coicsr`	0.	0.	0.	0.	0.
`distribute`	0.002	0.004	0.002	0.002	0.002
`analysis`	0.	0.	0.	0.	0.
`facto`	0.	0.	0.	0.	0.
`gmres/solve`	12.8	6.89	5.92	3.56	2.12
`ITER`	*29.9*	14.8	10.2	5.97	3.58

Table 2: Timing in seconds for the third iteration (reference scenario model 302, `out_loop3` file)

The configuration on only one node (first column of the left table) is pathological. It is strongly penalized by a very large part of the available memory occupied by JOREK processes (memory shortfall and system starts swapping). That's better to exclude this configuration in a performance analysis.

In the left table, the timers for an iteration that includes analysis and factorisation steps is presented. Two steps scale quite well: `construct_matrix` and `factorisation`. Three steps do not scale at all: `coicsr`, `distribute`, `analysis`. A step is in the middle with some scaling but not a good one: `gmres/solve` step. For one complete iteration (`ITER` keyword), the scaling is not so bad, this is partly due to the large relative weight of the `factorisation` step.

In the right table, the timers for an iteration that does not include analysis and factorisation steps is shown. The `distribute` step takes less time than previous table because only the RHS vector b is transferred among cores, not the whole matrix A. The execution time for one iteration is dominated by the steps `construct_matrix` and `gmres/solve` and scales reasonably well.

4 Conclusion

A Non Regression Tool has been installed in JOREK code that allows the user to check if its runs give the same results as compared to some reference scenarii. Three reference scenarii have been designed for models 199, 302 and 303. The comparison of one run against a reference run is performed thanks to a specific metric encasulated into a shell script. This metric is based on the numerical differences between several energy Fourier modes growth rates during a given time interval against one reference scenario.

The reference scenarii can also be used for benchmarking issues. We have compared in different configurations some timers outputted by the JOREK code. These timers give clues to determine a performant configuration when porting the code on new systems.

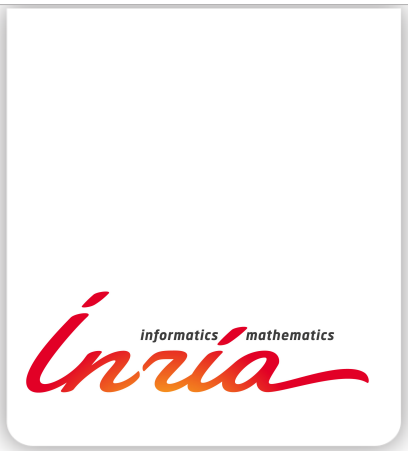
References

[CH08] Olivier Czarny and Guido Huysmans. Bezier surfaces and finite elements for MHD simulations. *J. Comput. Phys.*, 227(16):7423–7445, August 2008.

[HC07] G.T.A. Huysmans and O. Czarny. Mhd stability in x-point geometry: simulation of elms. *Nuclear Fusion*, 47(7):659, 2007.

[HMH+12] M. Hoelzl, P. Merkel, G. T. A. Huysmans, E. Nardon, E. Strumberger, R. McAdams, I. Chapman, S. Guenter, and K. Lackner. Coupling JOREK and STARWALL for Non-linear Resistive-wall Simulations. *Theory Of Fusion Plasmas, Varenna Workshop proceedings*, 0(0), 2012.

[NBHC07] E. Nardon, M. Becoulet, G. Huysmans, and O. Czarny. Magnetohydrodynamics modelling of H-mode plasma response to external resonant magnetic perturbations. *Physics of Plasmas*, 14(9):092501, 2007.



RESEARCH CENTRE
BORDEAUX – SUD-OUEST

351, Cours de la Libération
Bâtiment A 29
33405 Talence Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399