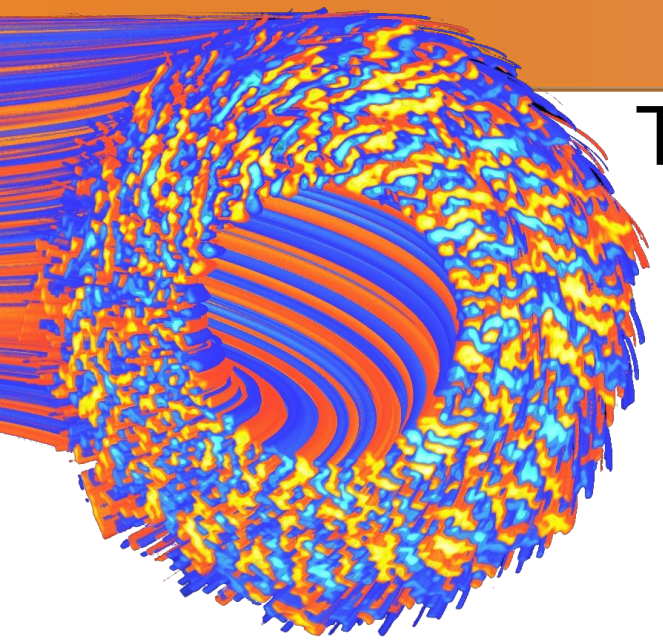
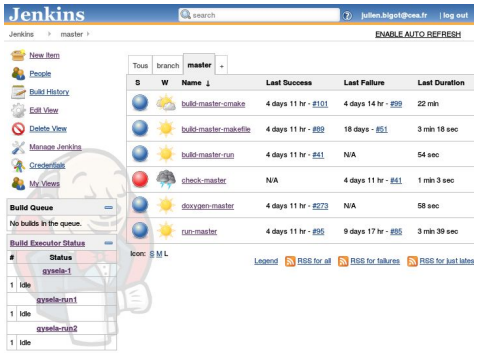
Continuous Integration



Towards more reproducible science the Gysela5D case



Julien Bigot[1]

In collaboration with:

- Chantal Passeron[2], Guillaume Latu[2]
- Virginie Grandgirard[2], Fabien Rozar[1,2]
- Laurent Léger[3], Isabelle Dupays[3], Marie Fle[3]

[1]Maison de la Simulation, [2]CEA/IRFM, [3]CNRS/IDRIS

Many thanks to the Inria Continuous Integration platform Team

C2S@Exa -- Bordeaux – Jul 10, 2014

Motivations

MAISON DE LA SIMULATION

- Improve confidence in simulation results
  - Is observed behavior due to a bug or is it a new physics discovery?
- Improve reproducibility of simulation results
  - What configuration was used to get some results?
  - What code changes impact the results?
- Improve code “cost”
  - Reduce crash during production
    - Reduce loss of computation hours
  - Reduce time lost in tracking bugs
    - Reduce loss of developer hours

The Gysela5D case

- Magnetic confined plasma simulation code
  - For study of transport & turbulences in tokamak
- Developed in Fortran95 (≃47kloc) & C (≃2,3kloc)
  - Alternative implementations for lots of modules
- About 10 years of history
  - CVS, then SVN
- ≃ 5 developers
  - applied mathematicians, computer scientists
- ≃ 5-10 users
  - Mostly physicists, some applied mathematicians
  - Run & read the code, propose code change

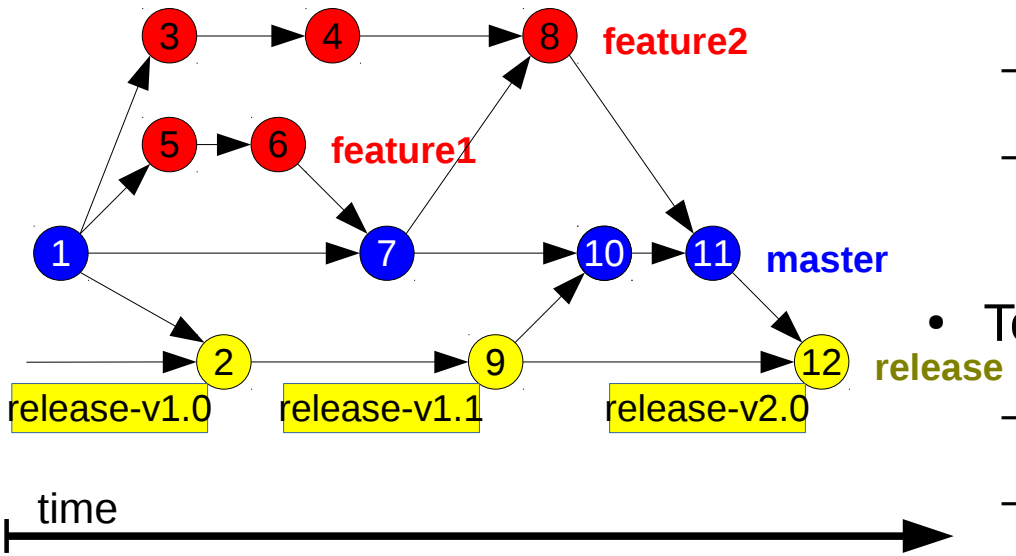
Outline

MAISON DE LA SIMULATION

- Context
- The platform
- Ensuring the code compiles & runs
- Validating the results
  - Bit exact comparison
  - What to do when it goes south?
  - Smarter comparison
- Conclusion & future work

The Gysela dev. work-flow

- Left SVN linear history (all changes at same level)
- A git based work-flow
  - A release branch (The production code, what users see => SVN)
  - An integration branch (“master”, a.k.a. next release)
  - Multiple features branches (concurrent developments)



- Achievement:
  - 1 change = well identified feature branch
  - Master should always be stable
    - Especially before release
- Tools:
  - Dedicated git commands
  - Gitlab for merge requests & bug/fix assoc.

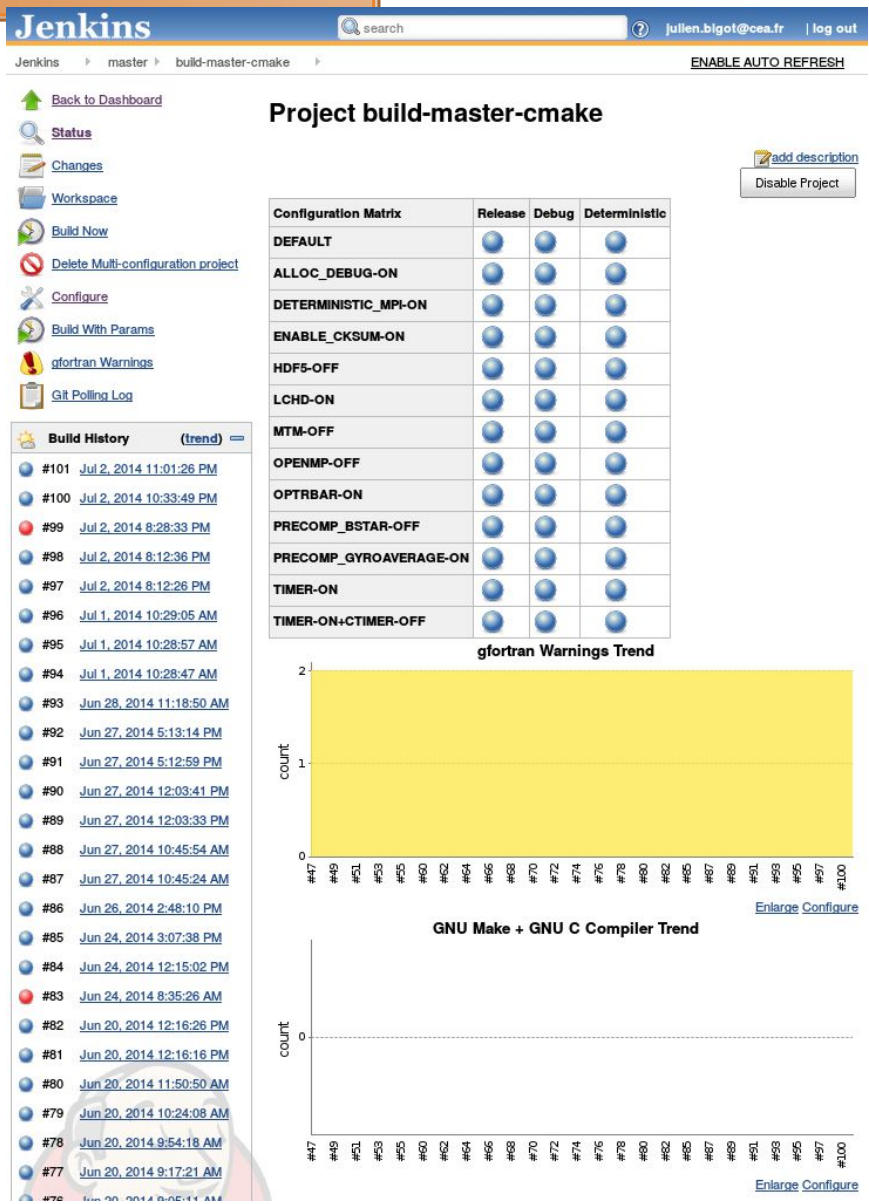
CI integration in work-flow

- Automatic application on master
  - Triggered by a git hook on each change
    - i.e. a feature integration
  - On breakage: mail to the culprit + the dev mailing list
  - Also triggers other automated processes
    - Doxygen doc generation
    - Some static analysis (case consistency, dead code, ...)
- Manual application on feature branches
  - Dedicated bash command
  - Sends mail to the requester only
  - Expected before merge in master

Ensure the code compiles

MAISON DE LA SIMULATION

- Inria continuous integration platform
  - Jenkins
  - CloudStack based testbed (VMs)
    - For us: Linux amd64 + gcc + mpich
- On code change
  - Compile w. large combination of compile flags / code path
  - Compiler warning analysis & reporting
- Last 100 results & logs kept



Ensure the code runs

MAISON DE LA SIMULATION

- Keep one version of the executable
  - default compilation options
- Reworked launching strategy
  - Synchronous mode (vs. batch)
  - Self-contained (vs. launch script + binary)
- Execute mini-runs on the VMs
  - In 9 different configurations
  - With 2 MPI process x 1 OMP thread
  - About 2 minute runs each

Results validation

MAISON DE LA SIMULATION

- ☺ Gysela is a fully deterministic code
  - Message handling order fixed
  - Pseudo-random number generation is deterministic (fixed seed)
- ☹ Foundations not fully deterministic
  - Compilers FP optimizations can alter results
    - Unrelated code changes can impact optimizations applied
  - MPI reductions have no fixed operations order
    - FP rounding approximations can vary

Achieving deterministic execution

MAISON DE LA SIMULATION

- A dedicated compilation mode
  - Compiler option disabling FP optimizations
    - Keep user specified operation order
    - Ensure result reproducibility
  - Dedicated implementation of MPI_Reduce
    - Not as efficient (memory or time) as the original

      but
    - Always apply reduction operations in the same order
- Achievement: bit perfect results reproducibility
  - Currently only on same machine/compiler
  - Can this be portable?

Bit exact comparison

MAISON DE LA SIMULATION

- When the “*code version*” didn't change
- Comparison of the main 5D data in checkpoint files
  - All other quantities based on it
  - Removed the ghost zones from the checkpoint
  - HDF5 files ⇒ Comparison w. h5diff
  - Small runs ⇒ small files (2 x 4MB)
- With previous code revision
  - Reference checkpoint files stored on the Inria forge
- Between different input data
  - e.g. 4 iter, 1 restart, 4 iter vs. 8 iter

When results don't match

MAISON DE LA SIMULATION

- Find out where things went south
  - Deterministic mode generates traces (dedicated fortan module)
  - Based on instrumentation in code: `DBG_CKSUM_PRINT(var)`
  - Generated trace file:

```
# file line
var = value
# file line
var = value
...
```

For scalars: the value
For arrays: a FNV-0 hash +size

  - On code modification
    - Filename, line might change

      But

    - Execution order should remain stable ⇒ (var, value) order conserved
  - Intrusive instrumentation to cover large code
    - A dedicated branch, automatically merged w. master by jenkins when possible

When results can't match

MAISON DE LA SIMULATION

- Some changes modify the results
  - New physics
  - Change in mathematical scheme
  - Change in algorithms/parallelization
  - …
- Some changes remain very small
  - Change in the "*code version*" minor (Major.minor)
  - Changes ignored at 10^{-14} threshold
- Even small changes can lead to large bit by bit change
  - e.g. a shift of the 5D function in one dimension

Smarter comparison

MAISON DE LA SIMULATION

- Identify comparison metrics
  - Macroscopic values that remain stable
  - Input from physicists needed
- Make the code less sensitive to perturbations
  - e.g. Added a HF filter applied periodically
- Create data with well known expected results

  but ...
- Will requires a parallel machine to run in a reasonable time

Conclusion & future work

MAISON DE LA SIMULATION

- Gysela is a deterministic code, with reproducible results in deterministic mode
- Automated: doc generation, branch merge, static analysis, compilation, run, bit exact comparison
- Not everything can be automated: also a merge request system
- Built tools to trace execution & identify divergences
- Still a lot to do
  - Unit tests: more developments & execution
  - Physics-based macroscopic comparisons
  - Comparison of timing
  - Validation on a large range of architecture / compilers

  ⇒ **Port to actual super-computers** (WIP w. IDRIS)
  - Apply to more codes (work stated on Hydro & Jorek)