Sustainable Software Development in the Ginkgo Library

Hartwig Anzt, Terry Cojean, Goran Flegar, Thomas Grützmacher, Pratik Nayak

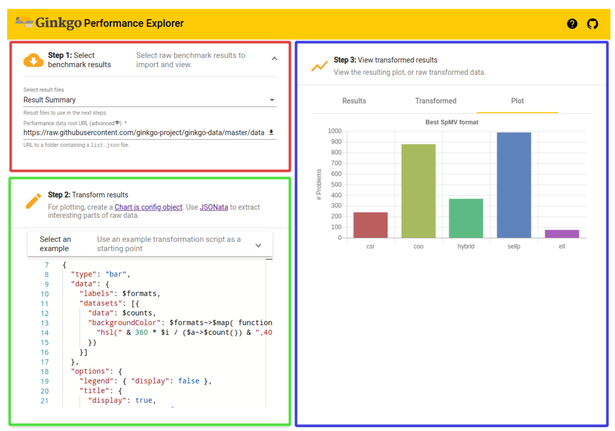
Code Contributions

The source code of the Ginkgo library can be accessed in a public git repository on GitHub. External developers are encouraged to contribute via pull requests. To preserve intellectual property of new developments, the public repository is mirrored into a repository hosted on GitLab that features private branches. To ensure high code quality, a list of development guidelines and tools are employed:

- **Contributor guidelines** list the conventions for code development.
- C++ concepts such as **smart pointers,** to prevent memory leaks, **runtime polymorphism** to enhance modularity, and **RAII** to enable efficient resource usage.
- The **readability** of the C++ code is enforced via automatically-invoked scripts based on **clang-format**.
- All functionality has to be complemented with **unit tests**.
- **Doxygen-ready documentation** of all new functionality is mandatory.
- All **code and dependencies are open source**.

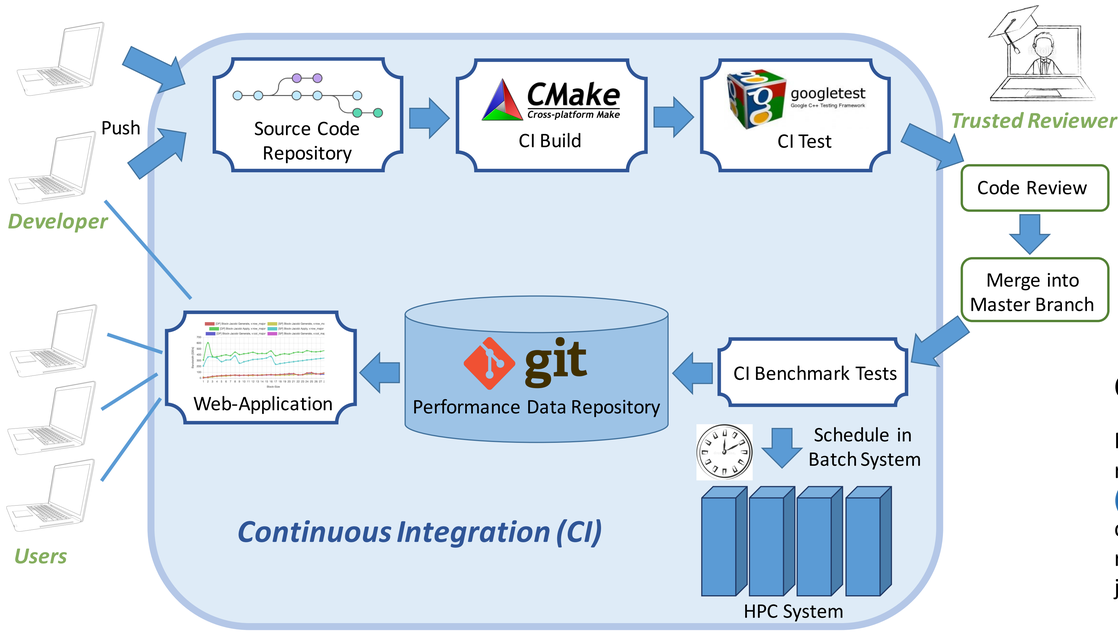
Interactive Performance Analysis

The **Ginkgo Performance Explorer**[5] (GPE) can be used to retrieve the performance data from the repository, and **visualizes** it in a **web browser**. GPE also implements an interface that allows users to **write scripts**, archived in a git repository, to extract particular data, compute particular metrics, and visualize them in many different formats (as specified by the script).



Software Interoperability

An important aspect of the Ginkgo development is the focus on **software interoperability**. We heavily leverage the expertise, code of conduct, and community guidelines offered by the **Extreme-scale Scientific Software Development Kit (xSDK[1])** and the **Better Scientific Software (BSSw[2])** initiatives. At the current point, we have full xSDK software compatibility, and expedite to soon become part of this effort. Furthermore, we hope that Ginkgo's focus on single-node execution allows for the smooth integration into larger software ecosystems like **Trilinos**[3].



Performance Result Database

The **performance results** of the benchmark runs are **archived** in a **publicly accessible** performance data repository based on git. Furthermore, the state of the machine, the environment, and even the compiled binaries are archived as GitLab artifacts. The intention of the CB system is to not only provide the developer with **feedback about the performance of the new functionality**, but also to **monitor the performance** of central functionality over time to **detect performance degradations**. Also **full reproducibility** of the runs is guaranteed. The raw data is accessible on GitHub which enables hassle-free visualization via the Ginkgo Performance Explorer (GPE).

Continuous Integration and Unit Tests

After each commit to the GitLab repository (or synchronization with the public repository), a **GitLab runner** is invoked to test the compilation and execution of the Ginkgo library (Continuous Integration, CI). The GitLab runner is executed on a private server where a list of **Docker images** generated via the **NVIDIA HPC Container Maker**[4] is used to simulate different execution environments in terms of Compilers and Third-Party Libraries. To test the correct execution, each functionality is complemented by **unit tests**. The unit testing is realized using the **Google Test** framework.

Review Process

If a pull request passes all build tests and unit tests, a **manual review process** is started. Two core developers have to approve the pull request before the new contribution is merged with the main branch of the Gingko repository. The review process ensures the **code integrity, sufficient documentation, and the coverage with unit tests**.

Continuous Benchmarking

Merging new functionality into the main branch of the Ginkgo repository automatically invokes a **Continuous Benchmarking (CB)** workflow. The CB system uses scripts to schedule pre-defined jobs on a specified HPC cluster and collects the results after the successful completion of the benchmark jobs.

References

[1]xSDK: https://xsdk.info/
[2]BSSw: https://bssw.io/
[3]Trilinos: https://trilinos.github.io/
[4]NHPCCM: https://github.com/NVIDIA/hpc-container-maker
[5]GPE: https://ginkgo-project.github.io/gpe/

Ginkgo

https://ginkgo-project.github.io/