UK Atomic Energy Authority

ExCALIBUR

Verification and Benchmarks Methodology

D3.5

The deliverable describes work for ExCALIBUR project NEPTUNE at Milestone 3.5. This is a description of how the verification and benchmarking of proxyapps has so far been discussed by work on project NEPTUNE. Verification here means checking that a piece of software meets its own internal design specifications ('solving the equations right' rather than 'solving the right equations'). There is overlap with the meaning of the term benchmarking, which is informally defined [1, § 5.3] to mean an inter-code comparison using 'a variety of solutions and performance metrics'. Work by the Oxford grantees to be delivered later has included the construction of analytic solution 'benchmarks', almost invariably of linearised equations for comparison with the output of their code. By contrast, the Milestone UKAEA report [2], the benchmarking exercise is directed much more towards understanding the behaviour of the hardware. In the UKAEA report on code structure and benchmarking [3], 'benchmarking' implies principally the performance aspect, specifically when code is refactored and/or additional features introduced. The inclusion of performance seems the best way to distinguish 'benchmarking' from 'verification' studies.

Work for project NEPTUNE on the topic of verification and benchmarking has almost inevitably been diffused among a number of other reports both by UKAEA staff [4], by the UKAEA report [5] wrapping the reports by Wright et al [6, 7] and other work by grantees [8]. The first report touches on the definition of proxyapps suitable for use in determining likely performance of the main application software. The wrapped reports by Wright et al are concerned with how to assess the likely variation in code performance among different platforms, again by using proxyapps. Metrics of portability are discussed in ref [6], and there is detailed discussion of programming language and hardware in both [6, 7]. Mentioned in ref [3, § 2] is the ExCALIBUR Benchmarking Working Group. This group is investigating use of ReFrame, which was conceived as a regression test framework for HPC systems and so presumably requires significant adaptation for benchmarking.

The last report listed [8], which is wrapped herein, covers a wide range of activities, so that only Section 4 is relevant to benchmarking. The recommendations made therein for the next step of NEPTUNE now follow. If verification and benchmarking are treated as separate activities, then the former may be achieved by a combination of GitHub Actions [9] with Docker images [8, § 4.1]. As to benchmarking, no single tool was identified as appropriate for NEPTUNE, and work to produce a suitable tool commenced with a specification [8, § 4.2]. The recommendation is that aspects of existing benchmarking software (TheMatrix, Devito and Gingko Performance Explorer, for references see [8, § 4.2.1]) be incorporated in a new NEPTUNE software for performance testing.

Acknowledgement

The support of the UK Meteorological Office and Strategic Priorities Fund is acknowledged.

References

[1] R.A. Bartlett, A. Dubey, X.S. Li, J.D. Moulton, J.M. Willenbring, and U.M. Yang. Testing of Scientific Software: Impacts on Research Credibility, Development Productivity, Maturation, and Sustainability. In J.C. Carver, N.P. Chue Hong, and G.K. Thiruvathukal, editors, *Software Engineering for Science*, pages 89–118. Chapman and Hall/CRC, 2017.

[2] L. Anton. Benchmarking requirements for NEPTUNE and available tools. Technical Report CD/EXCALIBUR-FMS/0016-1.00-M3.5.1, UKAEA, 2020.

[3] J. Parker, E. Threlfall, W. Arter, and W. Saunders. Code coupling and benchmarking. Technical Report CD/EXCALIBUR-FMS/0053-M7.2, UKAEA, 2021.

[4] E. Threlfall, J. Parker, and W. Arter. Design patterns evaluation report. Technical Report CD/EXCALIBUR-FMS/0026-M3.3.3, UKAEA, 2020.

[5] J. Parker and W. Arter. Domain-Specific Language (DSL) and Performance Portability Assessment. Technical Report CD/EXCALIBUR-FMS/0049-D3.2, UKAEA, 2021.

[6] S. Wright, B. Dudson, P. Hill, D. Dickinson, and G. Mudalige. Identification of Testbed Platforms and Applications. Technical Report 2047358-TN-02, UKAEA Project Neptune, 2021.

[7] S. Wright, B. Dudson, P. Hill, D. Dickinson, and G. Mudalige. Evaluation of Approaches to Performance Portability. Technical Report 2047358-TN-03, UKAEA Project Neptune, 2021.

[8] B. Dudson, P. Hill, E. Higgins, D. Dickinson, S. Wright, and D. Moxey. Task 0.1 Environments. Technical Report 2047356-TN-01-2, UKAEA Project Neptune, 2021.

[9] GitHub Actions. `https://docs.github.com/en/actions`, 2021. Accessed: September 2021.

UKAEA REFERENCE AND APPROVAL SHEET

	Client Reference:	
	UKAEA Reference:	CD/EXCALIBUR-FMS/0050
	Issue:	1.00
	Date:	September 30, 2021
Project Name: ExCALIBUR Fusion Modelling System		

	Name and Department	Signature	Date
Prepared By:	Wayne Arter BD	N/A	September 30, 2021
Reviewed By:	Rob Akers Advanced Computing Dept. Manager		September 30, 2021
Approved By:	Rob Akers Advanced Computing Dept. Manager		September 30, 2021

T/NA083/20
Fluid Referent Models

Task 0.1 Environment

Ben Dudson, Peter Hill, Ed Higgins, David Dickinson, and Steven Wright

University of York

David Moxey

University of Exeter

March 29, 2021

Contents

1 Executive summary

This document describes activities from January to March 2021, to coordinate community activities under the ExCALIBUR-Neptune project. The deliverable for this work package was to set up the development environment (version

control, continuous development/integration, automated testing and documentation services, coding standards). To achieve this we have set up development services, organised and attended community activities. The other part of this work package is to set up a testing framework for evaluating parallel scaling on e.g. Archer2 / Viking / Bede. In this period we have carried out a search of existing tools, and created an outline of the Neptune performance testing system design. This system will be implemented in task 0.2 (2021/22).

2 Development environment

2.1 Slack

We set up a Slack (`https://slack.com/intl/en-gb/`) work-space, `excalibur-neptune`, as a discussion and communication tool across all of the groups involved in ExCALIBUR-NEPTUNE. We considered alternatives such as Zulip (`https://zulip.com/`), which has similar features of asynchronous text messaging, group private messages, separate topics or channels, and file sharing. Zulip has some advantages over Slack, most notably it is free to host one's own instance (or rather, "free as in 'puppy'": the product is free, but there are still costs involved in the setup, maintenance and running of the service on one's own infrastructure). The main reason for choosing Slack over Zulip was the degree of familiarity people are likely to have with Slack: Zulip has a different model of conversations, somewhat similar to email, while Slack has a more traditional "chatroom" model. Many researchers are already members of at least one Slack workspace, so there is little incremental cost and a shallow learning curve to joining an additional workspace.

2.2 GitHub Organisation

Working with Research Software Engineers (RSEs) at UKAEA, we set up the GitHub organisation ExCALIBUR-NEPTUNE, creating two repositories:

- `Documents` – private repository archiving non-public documents, such as bid documents and reports.

- `Neptune` – public repository to collate Neptune components and documentation

We are managing access to these repositories as well as to the organisation as whole, inviting members of other groups to become organisation members, giving them permissions to create and manage their own repositories under the organisation. Through this mechanism, as of the start of March 2021, two proxy-apps (minepoch and Nektar-driftwave) are hosted under the ExCALIBUR-NEPTUNE organisation.

2.3 ReadTheDocs

We have set up ReadTheDocs (`https://readthedocs.org/`) to automatically build and host documentation in the `Neptune` repository. ReadTheDocs is built on Sphinx (`https://www.sphinx-doc.org/en/master/`) which reads and parses ReStructuredText (`http://docutils.sourceforge.net/rst.html`) files.

3 Community building

3.1 Hackathons

3.1.1 VECMA

The VECMA Hackathon ran from the 19th to 22nd of January 2021, and served as an introduction to the VECMA toolkit of uncertainty quantification (UQ) tools. We used two BOUT++ models, a simple 1D heat conduction model and a more complicated 2D "blob" model, as bases with which to learn the EasyVVUQ (`https://easyvvuq.readthedocs.io/en/dev/`) tool. We gained experience with using EasyVVUQ for BOUT++, and started understanding some of the challenges UQ is going to present to the ExCALIBUR-NEPTUNE project. During the Hackathon, we had conversations with the UQ group and fed our findings back to them.

In order to use EasyVVUQ with BOUT++, we had to first write a custom "encoder" and "decoder" in Python. The encoder turns a Python `dict` of input values into an input file that BOUT++ can read, while the decoder reads BOUT++ output file(s) into a Python `dict`. This was trivial to implement given BOUT++'s existing pre- and post-processing tools.

We next followed existing EasyVVUQ examples and tutorials to get a basic UQ workflow setup using the simple 1D model. These were easy to follow and to adapt to our model, and were greatly helped by access to EasyVVUQ developers and experts during the hackathon.

The 1D heat conduction model evolves the following equation in time, t:

$$\frac{\partial T}{\partial t} = \nabla_{||}(\chi \partial_{||} T) \tag{1}$$

where T is the temperature and χ is heat conductivity. BOUT++ grids are always 3D, even if some dimensions only have a single point. Here, we use 100 points in y, the parallel direction, and 1 point in both x and z. The initial condition is given by a Gaussian in y:

$$T(t=0) = A \exp[-(y-y_0)^2/(2w^2)]/(w\sqrt{2\pi}) \tag{2}$$

where A is the amplitude, y_0 the Gaussian centre, and w is the width of the Gaussian. Thus, we have four input parameters: χ, A, y_0 and w.

Overall, EasyVVUQ proved easy to do basic UQ and get results out. We initially used Polynomial Chaos Expansion (PCE) with this model, varying just χ and A, and measuring $T(y, t = 10)$. This model takes only a few seconds to run, and using 3rd order PCE resulted in 16 simulations, taking only a couple of minutes total. The PCE tools in EasyVVUQ have built-in tools for plotting the moments and Sobol indices, making simple analyses trivial. However, even using this simple model immediately uncovered some subtleties: it is (currently) not possible to give EasyVVUQ more information about the expected distribution. We know that T must always be positive, but when varying χ over multiple orders of magnitude, some simulations see T very quickly go to zero. The distribution of simulations can be heavily weighted close to zero, and the

resulting distribution of T computed by the PCE analysis can have significant amplitude at negative T, which is nonphysical. Similar difficulties are anticipated in any system where uncertainty in an independent or input variable varies over multiple orders of magnitude. One solution is to instead measure $\ln(T)$ instead of T, which enforces the positivity condition, but at the expense of making the resulting uncertainties more difficult to interpret. A more robust solution would be for EasyVVUQ to be able to incorporate additional *a priori* knowledge of the dependent variables.

The "blob2D" model is more complex, having spatial variation in two dimensions (x and z), and two evolving variables, the vorticity ω and the plasma density n:

$$\frac{\partial \omega}{\partial t} = -\left[\phi, n\right] + 2\frac{\partial n}{\partial z}\frac{\rho_s}{R_c} + D_n \nabla_\perp^2 n \tag{3}$$

$$\frac{\partial n}{\partial t} = -\left[\phi, \omega\right] + 2\frac{\partial n}{\partial z}\frac{\rho_s}{n R_c} + D_\omega \nabla_\perp^2 \omega \frac{1}{n} \tag{4}$$

$$\omega = \nabla^2 \phi \tag{5}$$

where ϕ is the electrostatic potential, $[\cdot, \cdot]$ is the Poisson bracket, $\rho_s = \sqrt{eT_{e0}m_i}/(eB_0/m_i)$ is the Bohm gyro-radius, e the electron charge, T_{e0} the initial electron temperature, m_i the ion mass, B_0 the magnetic field, R_c is the radius of curvature, D_n is the density diffusion coefficient, and D_ω is the viscous diffusion coefficient. This model is significantly more complex than the conduction model with many more input parameters, and takes several minutes to complete 50 timesteps on 16 cores. We used this model to investigate using EasyVVUQ on expensive turbulence models, varying the initial amplitudes of T_{e0}, the scale of the initial density perturbation, n_0, D_n, and D_ω. Because the output of this model is 2D in space, plus time, we used a number of lower-dimensional diagnostics as measurements instead. These consisted of the (x, z) position of the both the peak density and its centre of mass, as well as the velocity of this point.

The first thing to note is that the PCE sampler requires $(N+1)^M$ samples, where N is the order of polynomials used, and M is the number of parameters being varied. For our model where we are varying $M = 4$ parameters and using $N = 3$ we need 256 simulations. This is prohibitive on a local machine,

but EasyVVUQ includes several mechanisms for running simulations in parallel, including on clusters with job/queue managers such as SLURM. One mechanism is via `dask` (`https://dask.org/`), which works best for simple parallelisation on a local machine, and especially for Python kernels. The `dask_jobqueue` (`https://jobqueue.dask.org/en/latest/`) package extends this to SLURM clusters, but this proved to be very difficult to use for MPI parallelised programs. The last mechanism, `ExecuteSLURM`, takes a template SLURM job script and replaces variables with concrete values, and offers some control over the number of jobs to submit at once. This turned out to the be the most robust of the three mechanisms tried, and although it did not offer much benefit over a hand-written parameter scan, it is likely to be of more use when using hierarchical sparse grid sampling, where the parameter scan is incrementally refined. One thing to note is that while EasyVVUQ can launch simulations in parallel, the decoding – reading the output of the simulations – still happens in serial. Therefore if there is any cost to the decoding, it is probably wise to have this done as part of the simulation.

Lastly, we also explored using stochastic collocation (SC) instead of PCE. There was some pain to this. While the EasyVVUQ sampling and analysis objects are easily swapped between these two methods, the later analysis and plotting of results differ significantly. This means that a workflow written for one method needs several changes in order to convert it to use the other. Mostly these differences are due to incomplete or unimplemented methods, and it is expected that these differences disappear as EasyVVUQ matures.

3.2 Workshops

3.2.1 Performance Analysis

This ExCALIBUR-affiliated performance analysis workshop (`https://tinyurl.com/performanceanalysis2021`) held online workshops on the 21st 22nd of January, and 18th Feb.

Most of this work has been done by Joseph Parker and John Omotani (CCFE), with input and discussions from the BOUT++ team at York and elsewhere. BOUT++, and in particular the STORM model, has been instrumented with a

number of tools including Intel VTune and Score-P. The results of these workshops are collected in a github repository (`https://github.com/boutproject/cs-performance-tuning-workshop`).

3.2.2 HPC Development using C++ and SYCL

SYCL is one of the potential technologies for developing performance portable software under ExCALIBUR-NEPTUNE. We therefore joined workshops organised by Codeplay on 7th Jan and 17th Feb 2021. This included tutorials, exercises, and help from Codeplay to install and use SYCL compilers.

3.2.3 Towards Exascale Simulation of Integrated Engineering Systems at Extreme Scales

21 – 22 January, 2021: This was an ExCALIBUR meeting, at which Ben Dudson gave a talk "Coupling Codes at Exascale for the ExCALIBUR UKAEA NEPTUNE Nuclear Fusion Project". That talk presented an overview of the challenges, some representative examples of integrated simulations and code coupling in fusion, and invited participation in and input into the Neptune project.

3.3 Cross-task coordination

We have given talks at the main Neptune events, including the kickoff meeting 14th Jan, and wrap-up on 16th March, and regular progress update meetings.

Regular meetings have also been held with Oxford/Warwick group, and the STFC preconditioners group. Separate meetings have also been held with groups to discuss PinT and UQ bids and activities, to consider how these fit into the Neptune work plan. We have also participated in and in some cases led discussions in the ExCALIBUR-Neptune Slack workspace, to coordinate with the other Neptune tasks.

4 Testing

4.1 Correctness testing

Since Github is used to host the Excalibur-Neptune code, we propose to use Github actions for correctness and regression testing, as well as enforcement of coding styles and simple static code analysis. These tests can be triggered on pushes or pull requests, and can be configured to block merging into main branches unless passed.

The approach used by Nektar++ appears promising, in which Docker images are built and then used to run the test suite. If a test fails, this means that the same docker image can be downloaded and run on a developer's machine. This helps reproduce and identify errors, which might otherwise only occur on the testing server.

Github actions runs on virtual servers, with typically one or two cores, and inconsistent performance. This makes it unsuitable for performance testing, for which a bespoke solution is being developed, described in the next section.

4.2 Performance testing

We have started writing the specification for a system for monitoring performance of ExCALIBUR-NEPTUNE components and proxy-/mini-apps (hereafter collectively "apps"). Some requirements:

- Can be run manually, but amenable to automation: we want to be able to track the performance history of a given app, while still maintaining the flexibility to run *ad hoc* experiments across different app and hardware configurations;
- Flexible output: the performance metrics that we want to measure and track may change over time or between apps or experiments. We don't want to define a rigid schema now only to need to continually change it later;
- App agnostic: ExCALIBUR-NEPTUNE will be made of many compo-

nents, with many proxy-apps developed along the way, and making use of a variety of performance profiling tools. We want a single performance testing framework that can handle all of this variation;

In order to satisfy the first requirement, we have chosen to start developing a "push" framework, where data is collected on a machine and pushed to the data repository, rather than a "pull" framework where a server launches jobs on remote machines and pulls the data. This leaves open the option of automating the performance testing and converting the "push" framework (at least partly) into a "pull" one.

Our proposed framework consists of several components:

- *Test configuration files*: define an individual run of an app
- *Runner*: reads *test configuration files* and runs an app
- *Performance data files*: output from an individual run of an app
- *Uploader*: collates performance data files and uploads them to the data repository
- *Data repository*: stores performance data files
- *Dashboard*: interprets and displays data from data repository

Test configuration files need to be human readable, as these will be written by humans. This rules out formats such as JSON, which are suitable for machine-machine transfer of data, but are not human-friendly. There are a variety of text file formats that would be suitable; we are currently looking at TOML (https://toml.io/en/). The schema of these configuration files is still a work in progress, but there are several requirements:

- App executable location
- App input file(s) location(s)
- Performance tool (optional)
- *Performance data file* output location

The runner would read the test configuration files, and launch the app, possibly wrapped or instrumented with a separate performance profiling tool. This set up would allow automatic scanning for configuration files, and so expanding the performance test suite could be done through simply adding a new file.

The performance data files should be structured text files of some form, most likely JSON, to facilitate interoperability. This gives us the most flexibility in terms of moving to more rigid schemas later on or databases.

The data repository will be a plain GitHub repository. The uploader can then be a simple wrapper around git.

4.2.1 Existing Solutions

TheMatrix (https://github.com/devitocodes/thematrix) is a similar project for the Devito (https://www.devitoproject.org) symbolic finite difference library. TheMatrix runs performance benchmarks of Devito on a variety of hardware hosted in the Azure cloud service, and uses Airspeed Velocity (https://asv.readthedocs.io/en/stable/) to visualise the results. While meeting several of the ExCALIBUR-NEPTUNE requirements, there are a few major downsides. Firstly, Airspeed Velocity is limited to profiling Python tools/kernels only. This is essentially a show-stopper for ExCALIBUR-NEPTUNE in terms of being able to monitor the performance of proxy-apps written in several different languages. The other significant point is that TheMatrix is built around running the tests on the Azure platform, which does not currently meet the needs of ExCALIBUR-NEPTUNE.

Another similar project is Gingko Performance Explorer (GPE) (https://ginkgo-project.github.io/gpe/), another performance monitoring solution tied to a particular numerical package, Gingko (https://ginkgo-project.github.io). GPE is designed to be run automatically as part of a Continuous Integration/Continuous Development (CI/CD) process. After a commit to the development branch of the Gingko project, a GitHub Actions runner starts GPE, which pushes jobs to HPC systems, where tests are run and the performance is measured. GPE then periodically "checks in" to the HPC to see if the jobs have finished, and if so, collates the results. A particularly interesting feature of GPE is the data visualisation, which allows custom queries to be written and visualised directly in a web browser.

From an initial survey it seems that there is no existing solution which meets all the needs of ExCALIBUR-NEPTUNE. Partial solutions exist, and aspects of these will be adopted, to inform the design of the Excalibur-Neptune system. Since we are designing this to be a generic tool, it is likely to also be of interest to a wider community.