2023-09-27

D3.5 – Report on the reduction of numerical precision for computation and I/O

Version 1.0

GA no 952165

Dissemination Level

- ☒ PU: Public
- ☐ PP: Restricted to other programme participants (including the Commission)
- ☐ RE: Restricted to a group specified by the consortium (including the Commission)
- ☐ CO: Confidential, only for members of the consortium (including the Commission)

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

Document Information

Project Title	Targeting Real Chemical accuracy at the EXascale
Project Acronym	TREX
Grant Agreement No	952165
Instrument	Call: H2020-INFRAEDI-2019-1
Topic	INFRAEDI-05-2020 Centres of Excellence in EXascale computing
Start Date of Project	01-10-2020
Duration of Project	36 Months
Project Website	https://trex-coe.eu/
Deliverable Number	D3.5
Deliverable title	D3.5 – Report on the reduction of numerical precision for computation and I/O, as seen in GA
Due Date	M36 – 2023-09-30 (from GA)
Actual Submission Date	2023-09-27
Work Package	WP3 – Engineering solutions for exascale
Lead Author (Org)	Pablo de Oliveira Castro (UVSQ)
Contributing Author(s) (Org)	François Coppens (UVSQ), Aurélien Delval (UVSQ), Éric Petit (INTEL)
Reviewers (Org)	Mariella Ippolito (CINECA), Ivan Stich (IPSAS)
Version	1.0
Dissemination level	PU
Nature	Report
Draft / final	Final
No. of pages including cover	51

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

Disclaimer

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

The content of this document does not represent the opinion of the European Union, and the European Union is not responsible for any use that might be made of such content.

Versioning and contribution history

Version	Date	Authors	Notes
1.0	2023-09-27	Pablo de Oliveira Castro (UVSQ)	First Official Release

Abbreviations

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

Table of Contents

Document Information ii

Disclaimer iii

Versioning and contribution history iv

Abbreviations v

Table of Contents vi

Executive Summary 1

1 Introduction 1

2 Verificarlo-CI 2
- 2.1 Monte Carlo Arithmetic with Verificarlo 2
- 2.2 Designing CI for numerical correctness 3
  - Verificarlo pipeline 3
  - Verificarlo CI pipeline 3
    - Probe system 5
    - Generating a CI workflow 5
    - Running tests 5
    - Test reports 5

3 Numerical optimization of QMCkl's SMWB kernel 6
- 3.1 The case for using Sherman-Morrison in QMC=Chem 6
- 3.2 The Sherman-Morrison formula 7
- 3.3 The iterative problem of SM in QMC=Chem 8
- 3.4 Numerical methods 9
  - A naïve approach to iterative Sherman-Morrison 10
  - Partial pivoting: Maponi's method for solving linear systems 11
  - Reordering the updates 12
  - Slagel's method of update splitting 13
  - All at once: the Woodbury matrix identity 14
- 3.5 HPC versions 16
    - Vectorisation 16
    - Zero-padding of arrays 16
    - Pointer aliasing 16
    - Contiguous memory and memory alignment 17
    - Manual loop-unrolling: 2×2 and 3×3 WB and the Blocking kernel 17
    - Explicit array dimensions and kernel-template generation 20
- 3.6 Usage of Verificarlo-CI during SMWB kernel development 21

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

3.7 Experiments with QMC=Chem ... 23
Dataset extraction ... 23
Problems in the current implementation ... 24
Issue 1: Numerical accuracy with large number of determinants ... 24
Issue 2: Low arithmetic intensity (HPC) ... 24
Measurement machine architecture and measurement procedure ... 25
Results for the numerical accuracy ... 27
329 α-determinants ... 27
Numerical drift ... 27
Numerical accuracy independent of number of updates ... 28
15784 α-determinants ... 28
Results for the performance ... 29
Performance per rank-1 update ... 29
Performance stratification ... 31
3.8 Classification of all the kernels ... 31
3.9 Final recommendations ... 32
Single rank-1 updates ... 32
All other cases ... 32

4 Impact of Mixed-precision in CHAMP ... 32
4.1 Verificarlo and the VPREC backend ... 32
4.2 Delta-Debug and VFC-DD ... 33
4.3 Methodology ... 33
4.4 Mixed precision experiments on CHAMP ... 34
Total energy ... 34
4.5 Inter-nuclear forces ... 35
4.6 Recommendations ... 36
Single Precision only mode ... 36
Try on different datasets ... 37
Move to Intel SVML High Accuracy for `pow` and `log` ... 37
Optimisations on forces probably marginal ... 37

5 Conclusion ... 38

References ... I

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

Executive Summary

This deliverable adresses the critical concern of floating-point accuracy in numerical simulations and computation-intensive codes, emphasizing the need for early detection and resolution of numerical bugs. It introduces Verificarlo-CI, a continuous integration workflow that monitors and optimizes numerical accuracy during code development within QMCkl. It also details the design of optimized versions of Sherman-Morrison-Woodbury (SMWB) kernels and show they achieve a good tradeoff between accuracy and performance. The report also explores the benefits of mixed precision in the CHAMP code. Overall, it underscores the importance of early numerical accuracy assessment and offers practical solutions for tuning code performance and accuracy.

1 Introduction

Floating-point accuracy is an important concern when developing numerical simulations or other computation-intensive codes. Real numbers cannot be represented exactly on a computer. For this reason, their implementation requires finite approximations such as the IEEE 754 floating-point (FP) representation that is widely used today on modern computers [1]. Due to the approximations, numerical simulations with a large number of operation are prone to instabilities. Identifying and fixing numerical bugs is essential to achieve high accuracy in simulation codes. Nevertheless, not all algorithms require the same precision. Less sensitive kernels can be adapted to operate on lower precisions, this can provide optimizations in terms of energy, computation time, data transfer time, and storage size.

Tracking the introduction of numerical regression is often delayed until it provokes unexpected bugs for the end-user, resulting in a tedious and costly process for the code developer to locate the issue. To facilitate the tracking of numerical bugs of TREX packages, we developed Verificarlo CI, a continuous integration workflow for the numerical optimization and debugging of a code over the course of its development. Section 2 presents the design of Verificarlo-CI. It is based on Verificarlo, a tool that extends the LLVM compiler to assess the accuracy of a code by using custom arithmetic models, in place of standard floating point operations. When integrated into a repository, it provides automatically updated test results and dynamic reports accessible through a Web browser.

Verificarlo-CI was build to monitor numerical accuracy within QMCkl, in particular, we used it during the development of the Sherman-Morrison-Woodbury kernels (SMWB). Given a matrix A and its inverse A^{-1}, SMWB kernels efficiently update the inverse after a series of rank-1 updates on A. They are an important building block in different Quantum Monte Carlo codes such as QMC=Chem and TurboRVB. Due to their importance, particular care was taken to optimize them both for numerical accuracy and performance. Multiple implementations have been developed offering different tradeoffs depending on the user's goals. In particular, some of the implementations reduce numerical precision in a controlled fashion to gain performance. Section 3 presents the design of SMWB kernels, characterizes their properties, and offers recommendations for the end user. The kernels have been released as part of QMCkl.

We conclude this report, in section 4 with an analysis of mixed-precision benefits for the code CHAMP. Using Verificarlo, we explore the sensibility of different CHAMP routines when executed

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

with single FP precision. Based on the report, we offer some recommendations.

Some experiments on reducing numerical precision for I/O storage were also conducted in collaboration with WP2. We tested how well the floating-point compression libraries developed by LLNL operate on TREX I/O HDF5 files. In particular, we tested the zfp library [2] lossless compression algorithms on the wave-function determinants of the CR2 dataset. Our prelimar experiments showed that the zfp library did not provide a significant advantage over standard compression algorithms such as GZIP on this dataset. Given that GZIP is shipped on the standard version of the HDF5 library, we decided to use GZIP.

2 Verificarlo-CI

Despite FP accuracy being a known issue, modern tools for software development do not provide automated numerical accuracy regression tests. To fill this need, we propose Verificarlo CI (or `vfc_ci`, for **V**eri**f**i**c**arlo **C**ontinuous **I**ntegration), a tool that complements the Verificarlo software and operates in the context of Git version management. GitHub and GitLab are very popular platforms to develop modern numerical libraries and applications, and both have features for continuous integration (CI), respectively GitHub Actions and GitLab CI/CD. These services are triggered on specific events such as merging a pull request. From there, a variety of tasks can be accomplished, such as running tests or opening issues. The integration of Verificarlo CI with these services, while being central to its design, is still optional, which means that the tool can be used independently, or be integrated in any other continuous integration workflow.

To facilitate its adoption, Verificarlo CI has been designed to be easy and fast to deploy, while still being flexible enough to be relevant for most applications. For this reason, the setup requires a few simple steps, and little to no attention afterwards. We provide the user with efficient ways to insert FP probes in their tests, execute them with Verificarlo, setup CI Actions, and finally access and interpret the results.

2.1 Monte Carlo Arithmetic with Verificarlo

Verificarlo [3, 4] is an open-source tool based on the LLVM compiler framework modifying at compilation each floating point operation with custom operators. After compilation, the program can be linked against various backends to explore FP related issues and optimizations.

To evaluate the numerical accuracy of a computation, Verificarlo computes the number of significant bits which is a measure of the relative numerical error. It captures the number of accurate bits in the FP mantissa against a chosen reference. Unfortunately for many complex programs or intermediate computations, an exact reference value is not known beforehand. To overcome this problem, Verificarlo uses Monte Carlo arithmetic (MCA), a stochastic method that can simulate numerical errors and estimate the number of significant bits directly.

MCA can simulate the effect of different FP precisions by operating at a virtual precision t. To model errors on a value x, MCA uses the noise function $\text{inexact}(x) = x + 2^{e_x - t}\xi$, where $e_x = \lfloor \log_2 |x| \rfloor + 1$ is the order of magnitude of x and ξ is a uniformly distributed random variable in the range $\left(-\frac{1}{2}, \frac{1}{2}\right)$. During the MCA run of a given program, the result of each FP operation is replaced

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

by a perturbed computation modeling the losses of accuracy [5]. From a set of MCA samples, it is possible to estimate the significant bits s_2 of a computation,

$$s_2 = -\log_2 \left| \frac{\sigma}{\mu} \right| \tag{1}$$

where σ and μ are respectively the standard deviation and mean of the samples. Sohier et al. [6] provide sound confidence intervals for s_2. Without making assumptions about the samples distributions, they show that with only 29 samples s_2 measures the number of significant bits with probability 0.9 and 0.95 confidence.

2.2 Designing CI for numerical correctness

Verificarlo pipeline

Figure 1 shows a bird's eye view of the Verificarlo pipeline. Verificarlo can be used in any language supported by the LLVM ecosystem such as C, C++, and Fortran. It has specialized compiler passes that replace all FP operations by callbacks. The compiler passes also collect contextual information that enables to locate numerical errors precisely in the code source and calling context. Doing the interposition at compiler level allows to take into account the compiler optimization effect on the generated FP operation flow. Furthermore, it allows to reduce the cost of this interposition by optimizing its integration with the original code.

The instrumented program calls a numerical backend depending on the desired analysis. Verificarlo includes six backends which are extensively documented in the user manual. The two most important backends are:

- the `mca-quad` backend replaces standard IEEE-754 computations by Monte Carlo arithmetic as described in the next section. To simulate MCA, intermediate operations need to be performed with higher precision. Native `float` operations are performed with a `double` precision and native `double` operations are performed with a `quad` precision.
- the `vprec` backend simulates the effect of using mixed-precision in a program. It allows to customize the size of the exponent and of the mantissa during FP operations. For a full description of this backend please refer to [7].

The latest version of Verificarlo fully supports OpenMP, Pthread and MPI parallel programs, and backends have been designed to be reentrant and keep a coherent state between threads of execution. Moreover, Verificarlo comes with multiple post processing tools, such as `vfc_ci`.

Verificarlo CI pipeline

It is on this basis that Verificarlo CI has been developed, with the goal to provide a fast and general way to automate such numerical accuracy tests. The idea is to maintain a separate Git branch in parallel to the *main* development branch in order to store the test results. This branch will be called the *CI branch*. Each time users push modifications to the main branch, a distant runner will execute a predefined set of tests and push the results to the CI branch. These files can easily be accessed

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

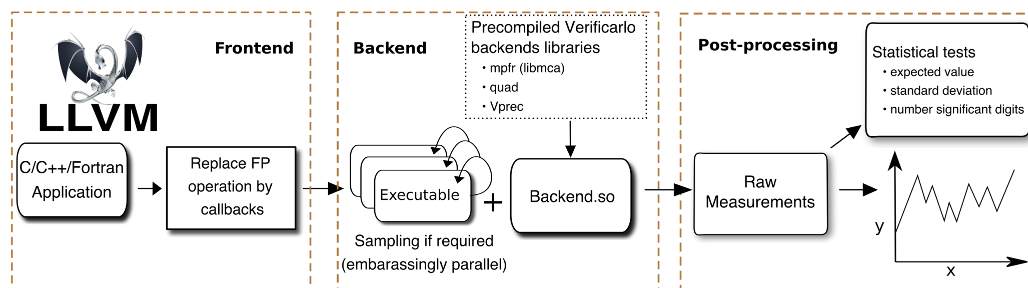

Figure 1: Bird's eye view of the Verificarlo pipeline.

by the users, who can then run a simple web server to dynamically access and visualize their results. This usage pipeline is shown in figure 2.

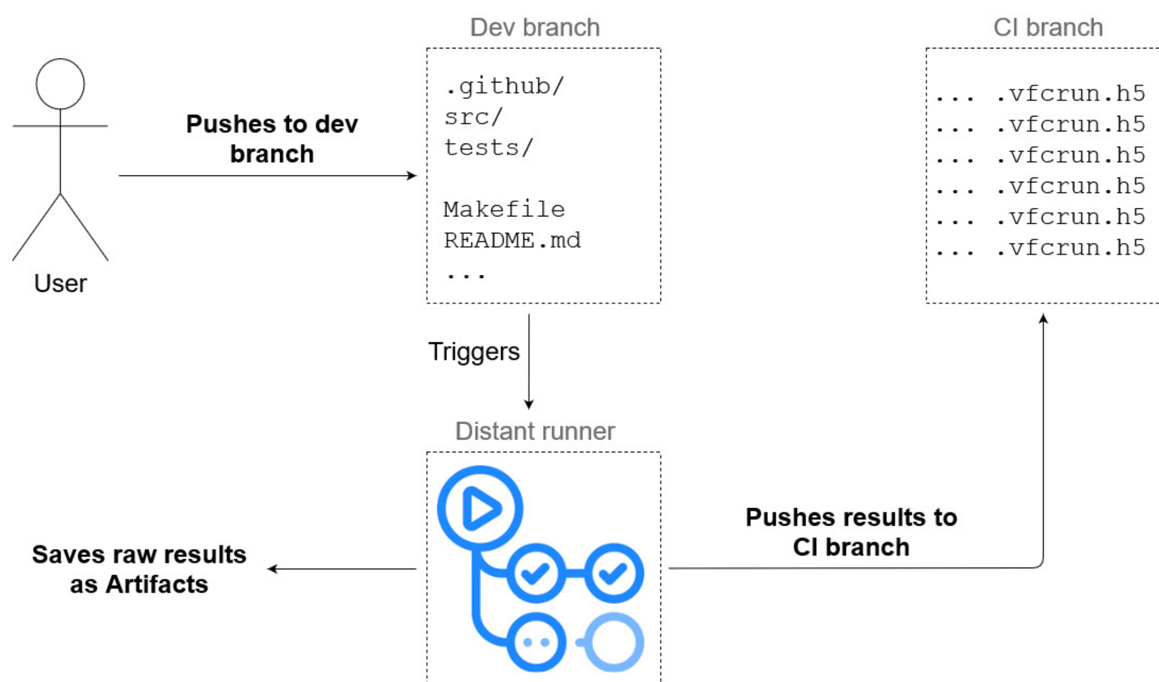

Figure 2: Verificarlo CI usage pipeline

All functionalities of Verificarlo CI are gathered into a single command-line interface (CLI) comprised of the three following sub-commands:

- `setup`: performs an automatic setup of the CI workflow, which can then be customised according to the user's needs.
- `test`: launches a test run by reading a configuration file that contains instructions to build and execute the tests.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

- serve: runs a server that reads test results files, and gives access to an interactive report to visualize results in a Web browser.

The following sections will describe the different steps required to operate this workflow.

Probe system To export variables directly from test programs, we design a source level C API which works by placing "probes" in a code. A probe is linked to both a test and a variable name, and the names used to register the probe will be the one displayed in the report. All probes will be stored in a single hash map structure. Exporting this structure to the vfc_ci tool is handled by the API. In order to use the probes system in Fortran, we provide an interface using the ISO_C_BINDING intrinsic module.

vfc_probe, vfc_probe_check and vfc_probe_check_relative are the main functions to register a new probe, with an optional accuracy target that can be placed on the variable. These targets can represent an absolute or relative error, and will be used to inform the user if the desired accuracy is not reached on some of their variables.

A probe is identified by a unique combination of test and variable name at vfc_ci's level. In the below example, the probe is identified by the "test"/"var" couple, and an absolute error threshold is set to 0.01:

```
vfc_probe_check(&probes, "test", "var", var, 0.01);
```

Generating a CI workflow To integrate Verificarlo CI with Github Actions or Gitlab CI/CD, the two branches structure described in section 2.1 will be used. The CI branch is an orphan branch. It facilitates the upload of the files from the CI runner, and the access by checking out the CI branch. The vfc_ci setup subcommand provides a convenient way to setup the workflow by creating the CI branch from a workflow configuration file on the main branch.

Running tests In order to be able to run the tests, the vfc_ci test subcommand requires a description of the tests and Verificarlo backends to run. It is specified in a JSON configuration file, which allows to specify and modify even complex test setups. Typically, the test subcommand will be launched by the CI workflow, but it can also be called manually at anytime.

All the data processing is done during this step: when using Verificarlo's MCA backend (or another non-deterministic backend), the tool computes the significant bits of variables and checks the target accuracy threshold to errors. The test results are exported to an HDF5 file, a hierarchical format commonly used in HPC applications. The HDF5 files can optionally embed the "raw" test results. In the default CI workflow, this raw data is stored as a job artifact and accessible for a limited time, to enable user defined additional analysis.

Finally, by default vfc_ci run links the test run to the last Git commit by fetching the associated git metadata to be saved.

Test reports vfc_ci uses the Bokeh Python module to generate plots inside an HTML report and make them available by running a server with the vfc_ci serve sub-command. Using widgets, the user can interact with the plots that are dynamically updated by requesting data from the server. The report is organised into different views:

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

- **Runs comparison**: this view compare the evolution of a variable over time (i.e. the different commits). Each metric is displayed in a dot plot where the different commits are shown on the x-axis. If an element of the plots is associated to a failing check, it will be displayed in red. Clicking on an element of the plots in this view will open the "Run inspection" view for the corresponding run.

- **Run inspection**: this view focuses on a particular run, and allows to visualize results for either one test, variable or backend, grouped by one of the two remaining factors. Currently, This view is only available for stochastic backends where s, μ and σ are defined.

- **Check table**: this view summarizes all the accuracy targets attached to probes allowing to check which ones are passing or failing.

Section 3.6 shows how Verificarlo-CI was used to detect bugs during the development of SMWB kernels within QMCkl.

3 Numerical optimization of QMCkl's SMWB kernel

This section addresses the numerical problems of the implementation of the Sherman-Morrison method within QMCkl. We validate the performance and accuracy using a benchmark from QMC=Chem.

This work addresses the following goals. To

- solve a numerical problem that exists in the Sherman-Morrison method, as implemented and used extensively in the QMC=Chem [8] software package,

- extend and generalise this method to include higher-rank cases (Woodbury matrix identity),

- implement these methods and optimise them to run efficiently on high performance computing architectures offering different precision-performance tradeoffs.

In section 2.1 we briefly explain why the Sherman-Morrison method is used in QMC=Chem. Then we formally introduce the Sherman-Morrison formula in Section 2.2. Finally, in Section 2.3 the problem with the current iterative implementation and how it arises will be explained as well.

3.1 The case for using Sherman-Morrison in QMC=Chem

In many QMC methods the many-body wave function $\Psi(\mathbf{r}_1, \dots, \mathbf{r}_N)$ for N electrons (where $\mathbf{r}_i$ is the location of electron $i \in \{1, \dots, N\}$) is expressed as an expansion of N_{det} determinants of Slater-matrices, containing ordered collections of single-electron basis-wave functions

$$\Psi(\mathbf{R}) = \sum_{k=1}^{N_{\text{det}}} c_k \, \det S_k^{\uparrow}(\mathbf{R}_{\uparrow}) \, \det S_k^{\downarrow}(\mathbf{R}_{\downarrow}) \tag{2}$$

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

where $\mathbf{R} = \mathbf{r}_1, \ldots, \mathbf{r}_N$, $\mathbf{R}_\uparrow$ and $\mathbf{R}_\downarrow$ are the subsets of coordinates associated with spin-up and spin-down electrons. The matrix elements of S_k^σ ($\sigma = \uparrow, \downarrow$) are defined as

$$[S_k^\sigma]_{ij} = \phi_j(\mathbf{r}_i) \tag{3}$$

and ϕ_j are the single-electron orbitals. The coefficients c_k take care of the normalisation of the many-body wave function ($\int_{-\infty}^{+\infty} |\Psi|^2 \mathrm{d}\mathbf{R}{=}1$).

Sometimes this expansion only contains one big Slater determinant, sometimes it contains many smaller ones. In our test-case, based on data extracted from QMC=Chem, it concerns the latter.

During the course of minimising the total energy of the system, in each Monte Carlo step the kinetic energy of the system

$$E_{\text{kin}} \propto -\sum_{i=1}^{N} \nabla^2_{\mathbf{r}_i} \Psi(\mathbf{r}_1, \ldots, \mathbf{r}_N) \tag{4}$$

needs to be calculated. Each Laplacian $\nabla^2_{\mathbf{r}_i}$ is acting only on the $\phi_j(\mathbf{r}_i)$ parts of the wave function. To compute $\nabla^2_{\mathbf{r}_i}\Psi$ for a given $\mathbf{r}_i$ the following identity is used extensively

$$\partial_{r_i} \det S = \det S \,\mathrm{Tr}\,(S^{-1}\partial_{r_i} S), \quad r_i \in \{x_i, y_i, z_i\} \tag{5}$$

We therefore need to keep track of the inverses of the Slater-matrices to compute basic quantities like the kinetic energy of the system.

Common manipulations of the many-body system to move towards a lower total energy are moving single electrons and/or manipulating electron orbitals. This corresponds to changes either in the rows or the columns in the Slater-matrix. Whenever there is a change in the Slater-matrix due to one these manipulations, the inverse of the Slater-matrix needs to be re-computed.

A full inverse can be computed with software libraries like LAPACK [9] or MKL [10], but it turns out that we do not need to recompute the entire inverse matrix. We can use the Sherman-Morrison formula to update the old inverse Slater-matrix using the changes of the Slater-matrix. This turns out to be much less computationally expensive in most cases than recomputing from scratch the whole inverse Slater-matrix from the updated Slater-Matrix.

3.2 The Sherman-Morrison formula

Any change in a single row or column of a general $N \times N$ invertible matrix A can be expressed by a matrix U, constructed by the matrix product of two vectors u and v

$$U = uv^\top \tag{6}$$

where $v^\top$ is the transpose of v. If there is a change in the mth $(1 \leq m \leq N)$ column of A, then v will be the mth column of the $N \times N$ identity matrix, while u contains the changes to the matrix

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

elements of A, such that the changed matrix $\widetilde{A}$

$$\widetilde{A} = A + U = A + uv^\top \tag{7}$$

A change like U is called a rank-1 update because the dimension of the image of U, when U is treated as an operator on a vector x, is 1

$$\dim(\mathrm{img}(U)) = 1 \tag{8}$$

Another way of seeing this is to realise that all the rows/columns of U are linear combinations of each other, and therefore the dimension of the row/column space of U is 1.

The new inverse $\widetilde{A}^{-1}$ can then be computed from the old inverse A^{-1} with the following formula

$$\widetilde{A}^{-1} = \left(A + uv^\top\right)^{-1} = A^{-1} - \frac{A^{-1}uv^\top A^{-1}}{1 + v^\top A^{-1}u} \tag{9}$$

This is in the literature called the Sherman-Morrison (SM) formula.

3.3 The iterative problem of SM in QMC=Chem

Multi-row or multi-column changes can be expressed by a rank-K matrix U, which is simply a sum of K rank-1 matrices U_k defined by

$$U = \sum_{k=1}^{K} U_k = \sum_{k=1}^{K} u_k v_k^\top \tag{10}$$

such that

$$\widetilde{A} = \sum_{k=0}^{K} A_k = A_0 + \sum_{k=1}^{K} u_k v_k^\top \tag{11}$$

where A_0 is identified with the unmodified starting matrix A.

The SM-formula can be applied iteratively for each U_k until all K changes to A^{-1} have been applied

$$A_k^{-1} = A_{k-1}^{-1} - \frac{A_{k-1}^{-1} u_k v_k^\top A_{k-1}^{-1}}{1 + v_k^\top A_{k-1}^{-1} u_k}, \quad 1 \leq k \leq K \tag{12}$$

and A_K^{-1} is simply the final $\widetilde{A}^{-1}$.

When K is small compared to the dimension N of the matrix (and N is not too small itself), computing the new inverse matrix using SM is much cheaper than inverting the whole matrix from scratch with routines like LAPACK that do a full LU-decomposition every time. However, the speed-up of applying a chain of K rank-1 updates with the SM comes with a cost.

The SM-method has a problematic property. When $K > 1$, a situation can arise where one of the U_k in the chain causes an A_k to become singular and $\det(A_k) = 0$. In that case A_k^{-1} is no longer defined, but when $\det(A_k)$ is close to zero, A_k^{-1} still exists, though it can have a large numerical error.

Even though the final inverse $\widetilde{A}^{-1}$ is guaranteed to exists, there is no such guarantee for the intermediaries A_k^{-1}. But there is a silver lining. Looking at Eqn. (11) we see that applying the updates to A is a commutative procedure; we may apply the update in whatever order we like.

Saying that $\det(A_k)$ zero is equivalent to saying that the denominator in Eq. (12) is zero. So we can introduce a 'break-down' parameter β such that

$$0 < \beta \ll 1 \tag{13}$$

where $\beta := 1 + v_k^\top A_{k-1}^{-1} u_k$. With this parameter we express what close to zero means. A value of $\beta = 1 \times 10^{-3}$ is customary in QMC=Chem. This values was also used during the experiments we did. Then for a given β, A_{k-1}^{-1} and u_k we stop updating when

$$\left|1 + v_k^\top A_{k-1}^{-1} u_k\right| < \beta. \tag{14}$$

When this happens update u_k is put in a waiting queue. When all the updates are applied in the first pass, the waiting queue is processed in a second pass. The same can happen in the second pass and we end up with a new queue. Most of the time this process ends and eventually all updates are applied in some random order. But sometimes the queue cannot be emptied and thus a good order does not exists. When this happens a full inverse is computed from scratch. It might be that if we would have started with another u_k we could have found a good order. But this kind of algorithm would probably be much slower than simple computing a full inverse.

In the next section we will introduce several numerical methods that will either mitigate this problem or remove the possibility for it to happen completely.

3.4 Numerical methods

In this section we list and explain the numerical methods and algorithms that we developed to solve the numerical problems present in the current implementation of iterative SM in QMC=Chem. They are not necessarily listed in the order that we developed and tested them, but rather in order of ascending success rate.

To measure the success rate we used a straight forward method. We can define the following residual matrix

$$\rho := \widetilde{A}^{-1}\widetilde{A} - I \tag{15}$$

we can use ρ to define an error on $\widetilde{A}^{-1}$. In the same way that we introduced the break-down parameter β before we define the following tolerance

$$0 < \tau \ll 1 \tag{16}$$

that will determine if the final $\widetilde{A}^{-1}$ is acceptable or not. Using the element-wise Max-norm we impose the following condition on the residual matrix ρ

$$\|\rho\|_{\max} < \tau \tag{17}$$

Whenever this inequality holds the numerical error on $\widetilde{A}^{-1}$ is acceptable. We have chosen value of $\tau = 1 \times 10^{-3}$, the same as the value of β. We choose this value because it was the typical order of magnitude of the norm of the residual matrix composed of the matrix/matrix-inverse pairs from the dataset extracted from QMC=Chem.

We now proceed to introduced the numerical methods we used and developed.

A naïve approach to iterative Sherman-Morrison

This method is the most simplistic of all the iterative methods and also the one that has the lowest success rate. It corresponds to Eq. (15) in P. Maponi's 2006 paper titled 'The solution of linear systems by using the Sherman-Morrison formula' [11]. This formula expresses the new inverse matrix A^{-1} as a product chain of intermediary updated inverse matrices $\{A_k^{-1}\}$

$$A^{-1} = \left(\mathbb{1}_N - \frac{A_{k-1}^{-1} u_k v_k^\top}{1 + v_k^\top A_{k-1}^{-1} u_k}\right) \cdots \left(\mathbb{1}_N - \frac{A_1^{-1} u_2 v_2^\top}{1 + v_2^\top A_1^{-1} u_2}\right) \left(\mathbb{1}_N - \frac{A_0^{-1} u_1 v_1^\top}{1 + v_1^\top A_0^{-1} u_1}\right) A_0^{-1} \tag{18}$$

During the course of applying the chain of rank-1 updates, the denominators in Eqn. (18) are compared with β as in Eqn. (13). When the condition is true the update process is stopped.

For a given number of updates K, this kernel works as follows:

Algorithm 1: The "Naïve" kernel

Data: A_0^{-1}, $\dim A_0^{-1}$, $\det A_0^{-1}$, K, $\{u_k, v_k\}$, β
Result: A_K^{-1}, $\det A_K^{-1}$
initialisation: loop counter: $k \leftarrow 1$;
while $k \leq K$ **do**
  compute: $c_k \leftarrow A_{k-1}^{-1} u_k$ (column vector);
  compute the denominator: $d_k \leftarrow 1 + v_k^\top c_k$;
  if $|d_k| < \beta$ **then**
    exit;
  update: $\det A_{k-1}^{-1} \leftarrow \det A_{k-1}^{-1} \times d_k$;
  select the row from A_{k-1}^{-1} that is updated: $e_k \leftarrow v_k^\top A_{k-1}^{-1}$ (row vector);
  update: $A_{k-1}^{-1} \leftarrow A_{k-1}^{-1} - c_k d_k^{-1} e_k$;
  increment loop counter: $k \leftarrow k + 1$;

Even though it is the kernel with the lowest success rate we decided include it for the reason that if only one update is considered, it will not fail and is the fastest of all the listed kernels here, due to it's low complexity.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

Partial pivoting: Maponi's method for solving linear systems

When we were first looking at the problem where a chosen order leads to a singular matrix in the course of applying a chain of rank-1 updates, we were guided by intuition to the notion of partial-pivoting. When performing Gaussian elimination on a matrix A to solve for the vector $\mathbf{x}$ in the matrix equation

$$A\mathbf{x} = \mathbf{b} \tag{19}$$

the choice of pivot matters. Making a bad choice for the pivot element can lead to large errors in the solution due to propagation and amplification of round-off errors. In partial pivoting the row with the largest absolute value is chosen to minimise these errors.

The work of P. Maponi [11] is a logical starting point when considering partial pivoting in the context of using the Sherman-Morrison formula. Instead of computing the inverse directly he is using the Sherman-Morrison formula iteratively to solve a system of linear equations. He does this by defining the intermediate solution vectors x_k and the auxiliary solution vectors $y_{k,l} = A_k^{-1} u_l$. The inverse is then updated using the formula

$$A^{-1} = \left(\mathbb{1}_N - \frac{y_{M-1,p(M)} v_{p(M)}^\top}{1 + v_{p(M)}^\top y_{M-1,p(M)}} \right) \cdots \left(\mathbb{1}_N - \frac{y_{1,p(2)} v_{p(2)}^\top}{1 + v_{p(2)}^\top y_{1,p(2)}} \right) \left(\mathbb{1}_N - \frac{y_{0,p(1)} v_{p(1)}^\top}{1 + v_{p(1)}^\top y_{0,p(1)}} \right) A_0^{-1}. \tag{20}$$

We tested Algorithm 3 from his work, which used the idea of re-ordering the updates explained in Section 3.4. For a given update l, first all vectors $y_{k,l}$ are evaluated. The $y_{k,l}$ that gives the largest value for $\left|1 + v_l^\top A_k^{-1} u_l\right|$ is chosen and applied. This kernel is referred to as **Maponi A3**.

We show later that the numerical and computational performance of this method is not on par with the other ones presented in this report. Perhaps because of the added complexity to first obtain the solution $\mathbf{x}$ via the auxiliary solutions $y_{k,l}$ and use the to reconstruct the intermediary $\{A_k^{-1}\}$ and A^{-1}.

For a given number of updates K, the kernel works as follows:

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

Algorithm 2: The "Maponi A3" kernel

Data: A_0^{-1}, $\dim A_0^{-1}$, K, $\{u_k, v_k\}$, β
Result: A_K^{-1}
for $k \leftarrow 1$ **to** K **do**
  compute initial auxiliary sequence $\{y_{0,k}\}$: $y_{0,k} \leftarrow A_0^{-1} u_k$;
for $l \leftarrow 1$ **to** $K-1$ **do**
  for a given l, select from $\{y_{l-1,k}\}$ the k that gives the largest value of:
  $d_k \leftarrow \left|1 + v_0^\top y_{l-1,k}\right|$;
  update: $A_{l-1}^{-1} \leftarrow A_{l-1}^{-1} - d_k^{-1} y_{l-1,k} v_k^\top A_{l-1}^{-1}$;
  for $k \leftarrow l+1$ **to** K **do**
    compute next auxiliary sequence $\{y_{l,k}\}$:
    $y_{l,k} \leftarrow y_{l-1,k} - \frac{v_l^\top y_{l-1,k}}{1+v_l^\top y_{l-1,l}} y_{l-1,l}, \quad k \in \{l+1, \dots, K\}$;

Reordering the updates

This is the approach currently used in QMC=Chem. It is an improvement of the previous "Naïve" method. When an update u_k causes Eqn. (14) to be true, instead of exiting it is kept in a delay-queue. The next update is then evaluated and if it does not trigger a break-down, it is applied and the inverse is updated. If not, it is also sent to the delay-queue. Then the next update is evaluated, etc.

When the whole list is evaluated the method considers the updates left in the delay-queue. If it is empty it is finished if it is not, the delay-queue is considered the new list of updates and they are re-evaluated. If one triggers a break-down it is again sent to a delay-queue. The whole algorithm is repeated until the update queue is empty.

Sometimes this is not possible and the number of updates left in the queue is equal to the number that were going in. In this case the algorithm exits with an error. It is then up to the user what to do next. The algorithm is implemented as a recursive function in C.

For a given number of updates K, the kernel works as follows

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

Algorithm 3: The "Reordering" kernel

Data: A_0^{-1}, $\dim A_0^{-1}$, $\det A_0^{-1}$, K, $\{u_k, v_k\}$, β
Result: A_K^{-1}, $\det A_K^{-1}$
initialisation:
loop counter: $k \leftarrow 1$;
number of later updates: $L \leftarrow 0$;
later updates queue: $\{u_l, v_l\} \leftarrow \varnothing$;
while $k \leq K$ **do**
  compute: $c_k \leftarrow A_{k-1}^{-1} u_k$ (column vector);
  compute the denominator: $d_k \leftarrow 1 + v_k^\top c_k$;
  if $|d_k| < \beta$ **then**
    add update u_k to the queue: $\{u_l, v_l\} \leftarrow u_k, v_k$;
    increment number of later updates: $L \leftarrow L + 1$;
    increment loop counter: $k \leftarrow k + 1$;
    continue to the next iteration;
  update: $\det A_{k-1}^{-1} \leftarrow \det A_{k-1}^{-1} \times d_k$;
  select the row from A_{k-1}^{-1} that is updated: $e_k \leftarrow v_k^\top A_{k-1}^{-1}$ (row vector);
  update: $A_{k-1}^{-1} \leftarrow A_{k-1}^{-1} - c_k d_k^{-1} e_k$;
  increment loop counter: $k \leftarrow k + 1$;
if $L = K$ **then**
  exit;
else if $L > 0$ **then**
  recursive call using A_{K-L}^{-1}, $\dim A_{K-L}^{-1}$, $\det A_{K-L}^{-1}$, L, $\{u_l, v_l\}$;

Slagel's method of update splitting

This method is an improvement overt the previous 'reordering' approach. It is inspired by Joseph Tanner Slagel's splitting method [12].

The improvement is simple and captured as a flow-chart in figure 3. Once an update u_k causes Eqn. (14) to be true, instead of only pushing the update to the delay-queue, the update is first split in half, $u_k \rightarrow \frac{1}{2} u_k$. Then the first half of u_k is applied since it can no longer cause Eqn. (14) to be true. Then the other half is pushed on the delay-queue. Then the next update is considered. Either it is applied entirely, or split in half. The algorithm is then executed recursively until the delay-queue is empty.
We also provide the details in algorithm 4. It can be formally proved that by using this method it will take a finite number of splits to apply any chain of rank-1 updates, as long as the fully updated matrix is guaranteed to have an inverse. If you are interested in the proof, it is found in Theorem 3.1.4 in Section 3.1.2 of [12].

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

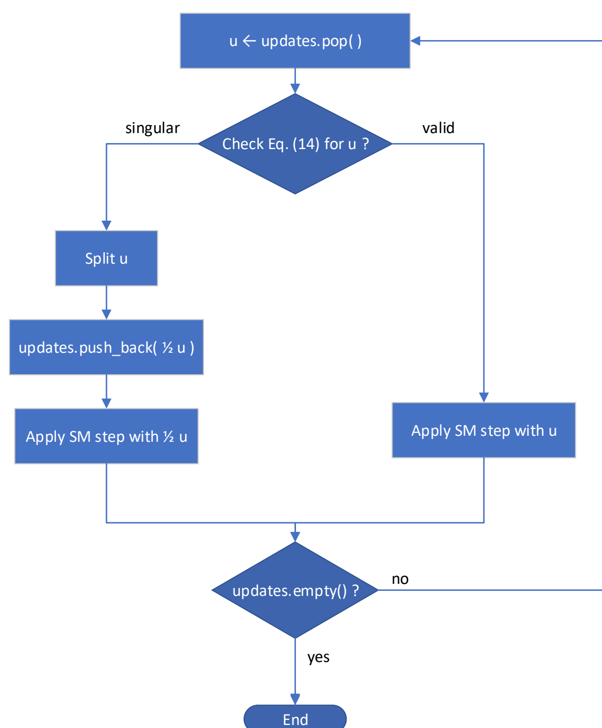

Figure 3: Flow-chart for update splitting

All at once: the Woodbury matrix identity

To circumvent the problem of iterative SM altogether we can use its generalised form, called the Woodbury (WB) Matrix Identity. The SM-formula is simply the special case for a rank-1 update matrix. This is its most general form

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \tag{21}$$

where $U : N \times K$, $C : K \times K$ and $V : K \times N$ are conformable matrices. The matrices U and V contain the updates and the update locations in the Slater-matrix repectively. Notice that

$$D := C^{-1} + VA^{-1}U \tag{22}$$

is at most $K \times K$. To give some perspective, when calculating the electronic structure and ground state properties of the compound Benzene with QMC=Chem, in 75% of the cases the number of updates that need to be applied to the inverse Slater-matrix is not larger than $K = 6$, whereas the Slater-matrix is $N = 21$.

Having said that, using Woodbury for rank-K updates can be more costly than K rank-1 SM-iterations, depending on the architecture and the implementation. There is no way to know a priory if this will be the case, so we recommend running a benchmark for an SM-kernel and a Woodbury-kernel and make the final choice based on the results.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

Algorithm 4: The "Splitting" kernel

Data: A_0^{-1}, $\dim A_0^{-1}$, $\det A_0^{-1}$, K, $\{u_k, v_k\}$, β
Result: A_K^{-1}, $\det A_K^{-1}$
initialisation:
loop counter: $k \leftarrow 1$;
number of later updates: $L \leftarrow 0$;
later updates queue: $\{u_l, v_l\} \leftarrow \varnothing$;
while $k \leq K$ **do**
    compute: $c_k \leftarrow A_{k-1}^{-1} u_k$ (column vector);
    compute the denominator: $d_k \leftarrow 1 + v_k^\top c_k$;
    if $|d_k| < \beta$ **then**
        split the current update in half: $c_k \leftarrow \frac{1}{2} c_k$;
        add half of update u_k to the queue: $\{u_l, v_l\} \leftarrow \frac{1}{2} u_k, v_k$;
        update the denominator: $d_k \leftarrow 1 + v_k^\top c_k$;
        increment number of later updates: $L \leftarrow L + 1$;
    update: $\det A_{k-1}^{-1} \leftarrow \det A_{k-1}^{-1} \times d_k$;
    select the row from A_{k-1}^{-1} that is updated: $e_k \leftarrow v_k^\top A_{k-1}^{-1}$ (row vector);
    update: $A_{k-1}^{-1} \leftarrow A_{k-1}^{-1} - c_k d_k^{-1} e_k$;
    increment loop counter: $k \leftarrow k + 1$;
if $L > 0$ **then**
    recursive call using A_{K-L}^{-1}, $\dim A_{K-L}^{-1}$, $\det A_{K-L}^{-1}$, L, $\{u_l, v_l\}$;

Notice also that for $C = I$, $U = u$ and $V = v^\top$, D reduces to

$$D = 1 + v^\top A^{-1} u \tag{23}$$

and since $v^\top A^{-1} u$ is scalar, we recover the SM-denominator $D^{-1} = 1/\left(1 + v^\top A^{-1} u\right)$.

In our implementations we use $C = I$, so the WB-formula we use to compute the new inverse is

$$A_{\text{new}}^{-1} = A_{\text{old}}^{-1} - B D^{-1} E \tag{24}$$

where

$$B := A_{\text{old}}^{-1} U \tag{25}$$
$$D := I + VB \tag{26}$$
$$E := V A_{\text{old}}^{-1} \tag{27}$$

The algorithm for K updates is listed below

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

Algorithm 5: The "$K \times K$ Woodbury" kernel

Data: A^{-1}_{old}, $\dim(A_{\text{old}})$, $\det(A_{\text{old}})$, K, U, V, β
Result: A^{-1}_{new}, $\det(A_{\text{new}})$
compute: $B \leftarrow A^{-1}_{\text{old}} U$;
compute: $D \leftarrow 1 + VB$;
compute: $E \leftarrow V A^{-1}_{\text{old}}$;
compute: $\det(D)$ from LU factorization;
if $|\det(D)| < \beta$ **then**
  exit;
update: $\det(A_{\text{new}}) \leftarrow \det(A_{\text{old}}) \times \det(D)$;
compute: D^{-1} from earlier LU factorization;
compute: $A^{-1}_{\text{new}} \leftarrow A^{-1}_{\text{old}} - BD^{-1}E$;

3.5 HPC versions

To run with optimum efficiency on HPC architectures we used several well known techniques. Most of the time was spent optimising the vectorisation that is discussed next. We also considered loop-unrolling for a fixed number of updates in the WB-kernel. To fine tune the performance even more and have specialised kernels for every system size we use source-code generation.

Vectorisation To vectorise the loops fully a few things need the be setup correctly simultaneously. Only then all the obstacles for the compiler are removed and it can do its work the way we want.

Zero-padding of arrays Loops in our SM kernel were not automatically vectorised completely when the leading dimension of an array was not an integer multiple of the CPU's vector register size. In this case a tail-loop was inserted and this has a negative impact on performance. It is then better to add a small amount of extra work by padding the arrays with extra zeros, so that their leading dimension becomes an integer multiple of the vector register size. Then tail-loops are unnecessary. Even though more work is done, vector-instructions are usually faster than scalar-instructions.

The amount of zero-padding depends on the size of vector registers of the CPU. For example for an Intel CPU with AVX2 extensions, the size of a vector register is 256 bits. This is equivalent to 8 ints/floats or 4 long ints/doubles 64 bit machine. For AVX512 these numbers double.

The datasets we used to benchmark the performance of our kernels were all zero-padded and in the argument list of the kernel calls there are always two arguments that need to be passed. The real dimension of the arrays and the zero-padded dimension, called the leading dimension.

Pointer aliasing The compiler is only able to vectorise the loops properly if it knows that the pointers that point to the arrays are pointing to distinct blocks of memory. In other words, the pointers cannot at any single point during the execution of the function body point to the same

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

memory address.

If the compiler is not sure of this and the memory of one of the arrays could be accidentally overwritten by using the pointers to other arrays, it will insert extra assembly instructions to make sure that the intended result is guaranteed. This will of course come with a performance cost so we need to prevent this from happening.

Luckily there is a way to tell the compiler that pointers do not alias by using the `__restrict` keyword in the pointer arguments of the function header. For example, for two `double`-pointers this could look like:

```
void some_function(double* __restrict DATA, double* __restrict OTHER_DATA);
```

Contiguous memory and memory alignment A final issue that can arise when vectorising loops is when the memory of the array is not contiguous and/or the data in the array is not aligned to memory addresses that are predictable. In this case simple pointer arithmetic is not possible and the compiler needs to insert extra assembly code to make sure it fetches the correct data. In that case vector instructions cannot be used and therefore the final code will only be partially vectorised, or not at all.

Again, we helped the compiler by making sure that the arrays contiguous in memory and that the data addresses is properly aligned to `n`-byte boundaries by declaring e.g., for a K element `double`-array `Array`:

```
double __attribute__( (aligned(8)) ) Array[K];
```

Manual loop-unrolling: 2×2 and 3×3 WB and the Blocking kernel For the Woodbury kernel we have improved the performance by considering special cases. Looking at Figure 6, it is evident that most of the time the number of updates that have to be processed is small, mostly one, two or three updates. It is for that reason that we made two special cases of the $K \times K$ Woodbury kernel: a 2×2 case and a 3×3 case. Because the number of rank-1 updates are known we can unroll and simplify some loops. It also permits us to use explicit expressions for computing the determinants and inverses of the 2×2 and 3×3 matrix D of Eq. (22), removing the need for expensive calls to LAPACK for the LU-decomposition and matrix inversion.

To deal with all the possible cases we made a kernel that combines the Splitting kernel with the WB 2×2 and 3×3 kernels that we call the "Blocking"-kernel.

To maximise arithmetic intensity the total number of updates is divided first in blocks of 3 rank-1 updates. Each of these blocks are then send to the WB 3×3-kernel. If any of these blocks fail due to $|\det D| < \beta$, the updates in the block are attempted with the Splitting-kernel and the split

updates are moved to a queue for later.

After all blocks are done we check if the remainder is either 2 or 1 rank-1 updates. If the remainder is 2, the updates are send to the WB 2×2-kernel. If this kernel fails due to $|\det D| < \beta$, the last 2 rank-1 updates are also attempted with the Splitting-kernel, any split updates are also moved to the queue for later.

If the remainder is 1, the last update attempted with Splitting-kernel, again adding half of it to the queue in case of a split.

Then finally, if there are any, the remaining halves in the queue are sent to the Splitting kernel which should always converge.

There is also a special case for 4 rank-1 updates. Performance measurements have shown that doing 4 rank-1 updates in 2 blocks of 2 rank-1 updates is faster than doing them in a block of 3 and the remaining one with the Splitting kernel. The method is the same as for the general case: if a block of 2 rank-1 updates fails, it is sent to the Splitting kernel and any split updates are moved to a queue to processed in the end.

The algorithm looks as follows:

Algorithm 6: The "Blocking" kernel

Data: A^{-1}, $\dim A^{-1}$, $\det A^{-1}$, K, U, V, β
Result: $\widetilde{A}$, $\det \widetilde{A}$
first initialisation:
number of later updates: $L \leftarrow 0$;
later updates queue: $\{u_l, v_l\} \leftarrow \varnothing$;
if $K = 4$ **then**
  call WB 2×2 with the **first** 2 rank-1 updates;
  if *WB* 2×2 *fails* **then**
    call Splitting-kernel[1] with the **first** 2 rank-1 updates;
    if *any splits* **then**
      add half of those updates u_k to the queue: $\{u_l, v_l\} \leftarrow \frac{1}{2}u_k, v_k$;
      increment number of later updates: $L \leftarrow L + (1 \vee 2)$;
  call WB 2×2 with the **last** 2 rank-1 updates;
  if *WB* 2×2 *fails* **then**
    call Splitting-kernel[1] with the **last** 2 rank-1 updates;
    if *any splits* **then**
      add half of those updates u_k to the queue: $\{u_l, v_l\} \leftarrow \frac{1}{2}u_k, v_k$;
      increment number of later updates: $L \leftarrow L + (1 \vee 2)$;
  if $L > 0$ **then**
    call Splitting-kernel[2] using A^{-1}_{K-L}, $\dim A^{-1}_{K-L}$, $\det A^{-1}_{K-L}$, L, $\{u_l, v_l\}$;
  exit;

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

second initialisation:
number of 3 rank-1 update blocks: $N \leftarrow \lfloor K/3 \rfloor$;
remainder: $R \leftarrow K \bmod 3$;
for *number of blocks N* **do**
    call WB 3×3 for each successive block of 3 rank-1 updates;
    if *WB 3×3 fails* **then**
        call Splitting-kernel[1] with the 3 rank-1 updates of the current block;
        if *any splits* **then**
            add half of those updates u_k to the queue: $\{u_l, v_l\} \leftarrow \frac{1}{2}u_k, v_k$;
            increment number of later updates: $L \leftarrow L + (1 \vee 2 \vee 3)$;
if $R = 2$ **then**
    call WB 2×2 with the **last** 2 rank-1 updates;
    if *WB 2×2 fails* **then**
        call Splitting-kernel[1] with the **last** 2 rank-1 updates;
        if *any splits* **then**
            add half of those updates u_k to the queue: $\{u_l, v_l\} \leftarrow \frac{1}{2}u_k, v_k$;
            increment number of later updates: $L \leftarrow L + (1 \vee 2)$;
else
    call Splitting-kernel[1] with the **last** rank-1 update;
    if *a split* **then**
        add half of the update u_k to the queue: $\{u_l, v_l\} \leftarrow \frac{1}{2}u_k, v_k$;
        increment number of later updates: $L \leftarrow L + 1$;
if $L > 0$ **then**
    call Splitting-kernel[2] using A^{-1}_{K-L}, $\dim A^{-1}_{K-L}$, $\det A^{-1}_{K-L}$, L, $\{u_l, v_l\}$;
exit;

Actually, only the non-recursive part of the Splitting kernel is used here. It doesn't call itself after it applied and/or split the updates. It only populates the update queue and increment the number of later updates if there are splits.
Here the full recursive version of the Splitting kernel is used.

Explicit array dimensions and kernel-template generation When the array dimensions and loop bounds are known to the compiler at compile time it can optimise more aggressively for loop-unrolling and vectorisation, greatly reduce the number of assembly instructions and increase the overall performance.

To that end a Python script is used to automatically generate C-source code for each specific system size. This allows the C-compiler to know always the array dimensions and loop-bounds at compile time. For the users a general kernel will be exposed in the QMCkl that contains a switch-mechanism that calls the internal specialised kernels that were generated by the Python script. A simple example of how this is done in C, for the fictitious kernel `kernel(size,`

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

other_parameters), is shown in Listing 1.

```
double kernel(size, other_parameters)
{
    switch(size)
    {
        case 1: return kernel_1(other_parameters);
        case 2: return kernel_2(other_parameters);
                         ...
                         ...
                         ...
        case n: return kernel_n(other_parameters);
    }
}
```

Listing 1: Automatic kernel selection based on the function argument size.

3.6 Usage of Verificarlo-CI during SMWB kernel development

In this section, we illustrate how we used Verificarlo CI during the SMWB kernel development process.

As shown previously, QMCkl implements different algorithms to apply SM formula with a set of updates (u_j, v_j), for $j = 1, ..., m$. The naive approach applies these updates in sequence. This method is called *Naive*.

Depending on the updates ordering, the SM denominator can be close to zero [13], meaning that the matrix A becomes singular. This makes the method numerically unstable. A refined algorithm using Slagel splitting [14] is called *Splitting*.

To implement the Woodbury formula, blocks of rank-3 and rank-2 updates are built. If the intermediate matrix update is singular, the corresponding updates will be applied as rank-1 with SM2. This method is called *Splitting+Blocking*. Because it changes the order of operations, one must ensure that the numerical accuracy is preserved compared to *Splitting*.

All these algorithms have been tested with Verificarlo CI on a set of matrices and updates from a real QMC=Chem use case on Benzene. For each test, we measure the residual with the squared norm $\sum_{i=1}^{n}(AA^{-1} - I)^2_{ii}$.

During the development of SMWB, the "Run inspection" view of the report allowed us to spot a bug that caused our implementation to numerically fail. Figure 4 shows a box plot obtained from this view: the number of significant bits s_2 has been computed for each combination of test and algorithm, and grouped by algorithms.

We observe some outputs for which SMWB has a high numerical error. After investigation, we discovered than in the initial implementation of SMWB, delayed updates were directly applied after each Woodbury step. This reduces the numerical stability because it increases the probability of producing singular intermediate matrices. It was fixed by moving all the updates to the very end of the update queue.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

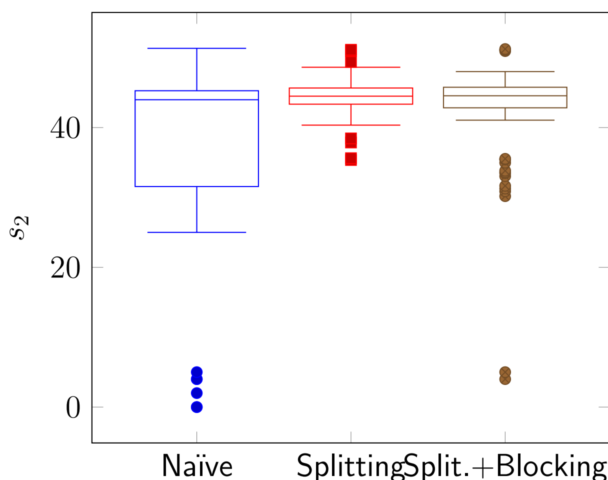

Figure 4: Significant bits of Frobenius norm for all cycles and algorithm combinations for commit `6f282f3`, grouped by algorithms. SMWB fails catastrophically in some cases.

Figure 5, obtained from the "Runs comparison" view, illustrates that from commit `67f5379`: after the fix, SMWB behaves similarly to SM2.

Thanks to Verificarlo CI we were able to detect this numerical discrepancy and fix it early in the development process.

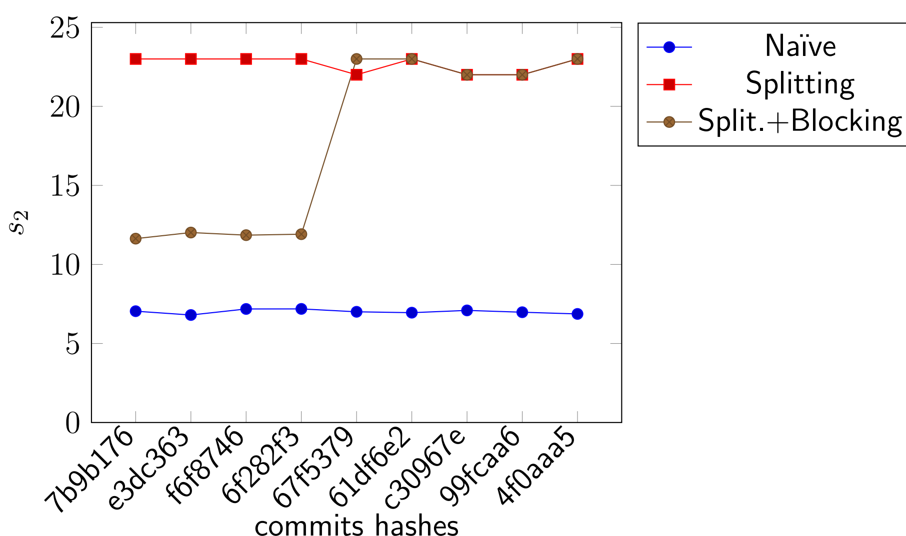

Figure 5: Significant bits of our different algorithms over commits for cycle 4263. SMWB's accuracy does improve.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

3.7 Experiments with QMC=Chem

QMC=Chem relies heavily on the use of SM for updating the determinants and inverses in the many-body wave function. So any gain in numerical accuracy and/or performance could have a significant impact on its overall performance.

Because we want to know how our algorithms would behave in QMC=Chem we tested them using real data extracted from a short run of QMC=Chem on the Benzene molecule. This way we can also account for the behaviour of the kernels due to any particularities of the data itself. How we extracted the dataset will be explained in the next section.

We then proceed with explaining the exiting problems with the current implementation of the SM-method. After that we elaborate on the measurement procedure and give details about the machine architecture on which the measurements have been performed. Finally, we present the experimental results.

Dataset extraction

We tested our kernels on datasets [15] that we extracted from QMC=Chem using Variational Monte Carlo (VMC) to run a ground state calculation for Benzene (42 electrons, 21 spin-up, 21 spin-down), using 329 and 15784 determinants respectively. The datasets are extracted after a single electron move, when the MOs and determiants in Eqn. 2 are being updated. Additional Fortran `write`-statements were added to the file `$QMCCHEM_ROOTsrc/det.irp.f` to extract and save the following data

- the number of the determinant in the chain, starting from 1 and omitting numbers $i \times N_{\text{dets}},\ i = 0, 1, \ldots$ that are computed with LAPACK
- the dimension of the Slater-matrix
- the number of rank-1 updates (= number of MO/column changes)
- the Slater-matrix S before update
- the inverse Slater-matrix before update
- the column-indices of the updates
- the updates themselves as replacement-updates

Fig. 6 shows the distribution of the occurrence frequency of the number of rank-1 updates. Both the datasets have a maximum of 15 rank-1 updates per changed determinant. The only difference is their relative occurrence frequency, for the 329-determinant dataset 1–6 rank-1 updates account for 75% of the cases, whereas for the 15784-determinant dataset it is 1–8 rank-1 updates.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

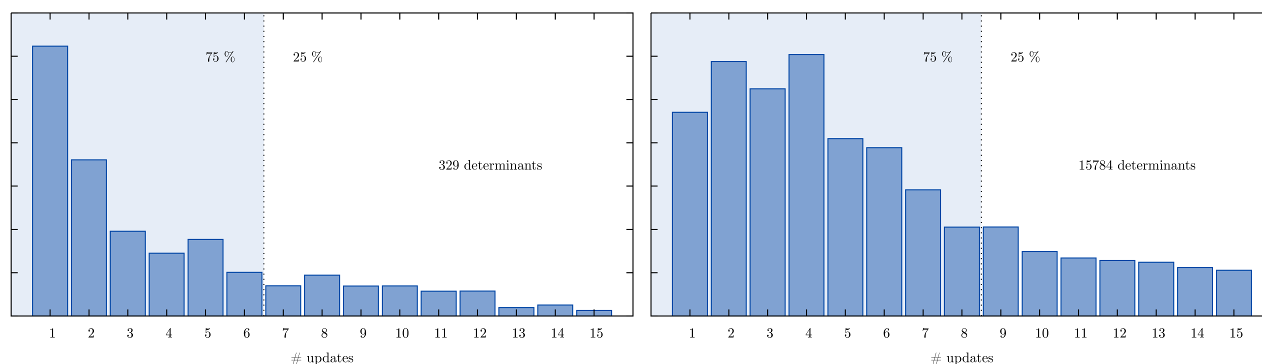

Figure 6: Distribution of the occurrence frequency for each number of rank-1 updates that can occur. The blue area represents the 3rd quartile of the distribution. For the 329-determinants case, 75% of the time updates consist of 1–6 rank-1 updates. For the 15784-determinants case updates consist of 1–8 rank-1 updates.

Problems in the current implementation

We now continue to list the issues in the current implementation of iterative Sherman-Morrison in QMC=Chem. All of these issues have been addressed in our improved kernels.

Issue 1: Numerical accuracy with large number of determinants The SM implemented in QMC=Chem at this moment is equivalent to the reordering method described in Section 3.4. In case there is one determinant along the chain that has more than one rank-1 update and a suitable order to apply the U_k cannot be found the whole inverse is recomputed with LAPACK. Then the next determinant in the chain is again updated using SM, if possible. If not, LAPACK, etc. In our particular case for Benzene with 329 determinants and the used value of $\beta = 1 \times 10^{-3}$ in QMC=Chem, LAPACK was never called mid-chain. But for the case of 15784 determinants, numerical accuracy suffers too much before the end of the chain is reached. LAPACK is called about 20 times per 15784-determinant chain.

Because of this, QMC=Chem is loosing more time during LAPACK calls then if it were to use only Sherman-Morrison. The challenge is therefore to first increase the numerical accuracy of the SM method. Once that is achieved, calls to LAPACK should only occur for the first determinant in the chain and thereby increasing the overall performance.

If we can improve the numerical accuracy of the kernels, calls to LAPACK mid-chain could be reduced as well as is part of its performance cost.

Issue 2: Low arithmetic intensity (HPC) All of the SM methods discussed below have one property in common in that they consist mostly of Matrix-Vector operations (BLAS Level 2). To increase the performance of applying multiple rank-1 updates even more it would be fruitful if we could

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

somehow increase the arithmetic intensity by using Matrix-Matrix operations (BLAS Level 3). One way of doing that is to use the more general Woodbury matrix identity discussed in Section 3.5 or 4.2.

Measurement machine architecture and measurement procedure

All the measurements have been performed on Intel-based PC with a Core i9-10900 64-bit CPU and 32 GB of non-ECC memory. The CPU has the following vector instruction extensions: SSE4.1, SSE4.2 and AVX2.

In the experiments with Benzene, the Slater-matrices will receive at most 15 rank-1 updates. One series of $1 \leq k \leq 15$ rank-1 is called an *update cycle*. For each update cycle we read the following data from the dataset (except D):

D : dimension of the Slater-matrix (only once at the beginning)

S : the current Slater-matrix

S^{-1} : the current inverse Slater-matrix

K : the number of rank-1 updates

C : the vector of length K containing the column numbers that need to be updated

U : $K \times D$ matrix containing the column updates

During the measurement, the following data is written to `STDOUT`

`CYCLE` cycle number

`UPDS` number of updates K in the cycle

`ERR_IN` check on the *input* matrices S and S^{-1}: 0 if $\|\rho_{\text{in}}\| < \tau$, 1 otherwise

`ERR_BREAK` check on the SM or WB denominator during update: 0 if $\left|1 + v_k^\top A_{k-1}^{-1} u_k\right| > \beta$, 1 otherwise

`ERR_OUT` check on the *output* matrices S and S^{-1}: 0 if $\|\rho_{\text{out}}\| < \tau$, 1 otherwise

`SPLITS` number of splits for kernels that use Slagel-splitting. *This is a global variable that is set to zero at the start and updated during kernel execution.*

`BLK_FAILS` number of failed blocks of 3- or 2 rank-1 updates for the Blocking kernel. *This is a global variable that is set to zero at the start and updated during kernel execution.*

`MAX` element-wise max-norm of ρ_{out}

`FROB` Frobenius-norm of ρ_{out}

`COND` Condition number $\|S^{-1}S\|$ of the output matrices

`CPU_CYC` Number of CPU cycles for the whole update cycle

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

CPU_CYC/UPD Number of CPU cycles / K

CUMUL Cumulative of all update cycles

REC Number of recursions for recursive kernels. *This is a global variable that is set to zero at the start and updated during kernel execution.*

Then the measurement program will do the following, based on *the chosen kernel* and N that has to be set at execution:

Set the global cumulator to 0;
for *each update cycle* **do**

1. Read S, S^{-1}, K, C and U from dataset
2. Check $||\rho_{\text{in}}|| < \tau$ and write to ERR_IN
3. Set CPU_CYC and SPLITS to 0

for N *repetitions* **do**

Make a copy S^{-1}_{copy} of S^{-1} and use it in the chosen kernel;
for *the chosen kernel* **do**

1. Poll the CPU cycle counter
2. Execute the chosen kernel and record the exit status in ERR_BREAK
3. Poll the CPU cycle counter again
4. Add CPU cycle difference to accumulator

1. Copy the updated S^{-1}_{copy} back to S^{-1}
2. Divide cycle- and split-accumulator by N and record in CPU_CYC and SPLITS
3. Add CPU_CYC to the global cumulator and record in CUMUL
4. Divide cycle-accumulator by K and record in CPU_CYC/UPD
5. Update S
6. Check $||\rho_{\text{out}}|| < \tau$ and write to ERR_OUT
7. Write the recorded measurements to STDOUT

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

Results for the numerical accuracy

To measure the numerical accuracy we use the two parameters β and τ defined in Eqns. (13, 16). It has been determined empirically through the use of QMC-Chem that a good "near zero" value, below which the quality of the updated inverse suffers too much, is $\beta = 1 \times 10^{-3}$. After a kernel has updated the inverse Slater-matrix the residual ρ is computed and the inequality Eqn. (17) is evaluated with $\tau = 1 \times 10^{-3}$. When the inequality holds the inverse Slater-matrix is considered numerically acceptable and PASSES. When the inequality doesn't hold, the inverse is considered numerically unacceptable and FAILS.

The number of PASSES and FAILS are collected and with them we define the following fail rate

$$\text{Fail rate} = \frac{\#\text{ of FAILS}}{\#\text{ of PASSES} + \#\text{ of FAILS}} \times 100\% \tag{28}$$

329 α-determinants

In Table 1 the fail rates are shown for 10384 update cycles and 329 determinants in the wave function. The Naïve-kernel is particularly bad because as the number of rank-1 updates increases along the 329-determinant chain the probability of encountering a rank-1 update that cause singular behaviour increases as well. The Reordering-kernel is already two orders of magnitude better, having only 26 failed updates. The Splitting– and Blocking-kernel are clearly the best, with only 21 failures. Unlike the failures for the Naïve-kernel, these are not failures of the kernels themselves due to denominators that are too small, but due to a too large max norm of the residual matrix ρ.

Kernel	# Pass	# Fail	Failrate (%)
Naïve	6598	3786	36.46
Reordering	10358	26	0.25
Splitting	10363	21	0.20
Blocking & Splitting	10363	21	0.20
Maponi A3	9544	840	8.09
Maponi A3 & Splitting	9921	463	4.46

Table 1: Fail rates for Benzene with 329 α-determinants and 10384 update cycles in total, $\beta = 1 \times 10^{-3}$ and $\tau = 1 \times 10^{-3}$.

Numerical drift For the case of 329-determinants there is a general trend that the max norm of the residual matrix $\|\rho\|_{\max}$ steadily increases while going along the chain, and then falls back again after the next first determinant is computed with LAPACK. This behaviour is shown in Fig. 7. This

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

suggest the numerical noise is slowly accumulated and carried along the chain. Unfortunately the failing cases are not in general situated near the ends of chains so it might be that they were simply caused by particularly ill conditioned input matrices.

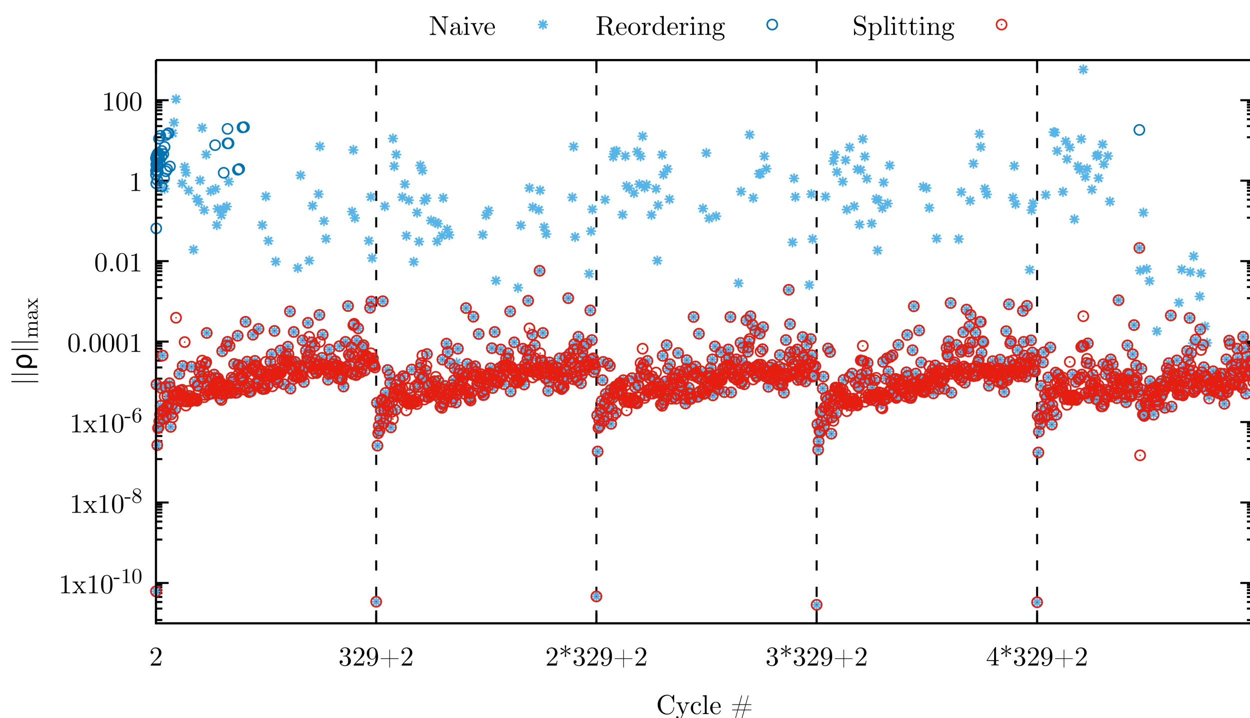

Figure 7: Numerical drift, steadily increasing from the beginning to the end of the 329 determinant update chain.

Numerical accuracy independent of number of updates Another interesting aspect of the SM and WB kernels is that the numerical accuracy, measured by the Max-norm of the residual matrix ρ does not seem to depend on the number of rank-1 updates they have to apply. This is shown in Figure 8. For most kernels and number of updates the Max-norm appears the be centred around 1×10^{-5}. There is one extra branch at 1 rank-1 update centred around 1×10^{-11}. These correspond to the second determinants in the chain as discussed just above.

15784 α-determinants

For this case we chose to show a slightly larger level of detail by giving the fail rates for five representative number of rank-1 updates: an overall fail rate *all*, the case where there is only *1* rank-1 update to compare against the Naïve kernel, *2* to compare against Woodbury 2×2 (WB2), *3* to compare against Woodbury 3×3 (WB3) and *6* to see how well the Blocking kernel performs against Reordering/Splitting. The results are summarised in Table 2. As in the 329-determinant case, the

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

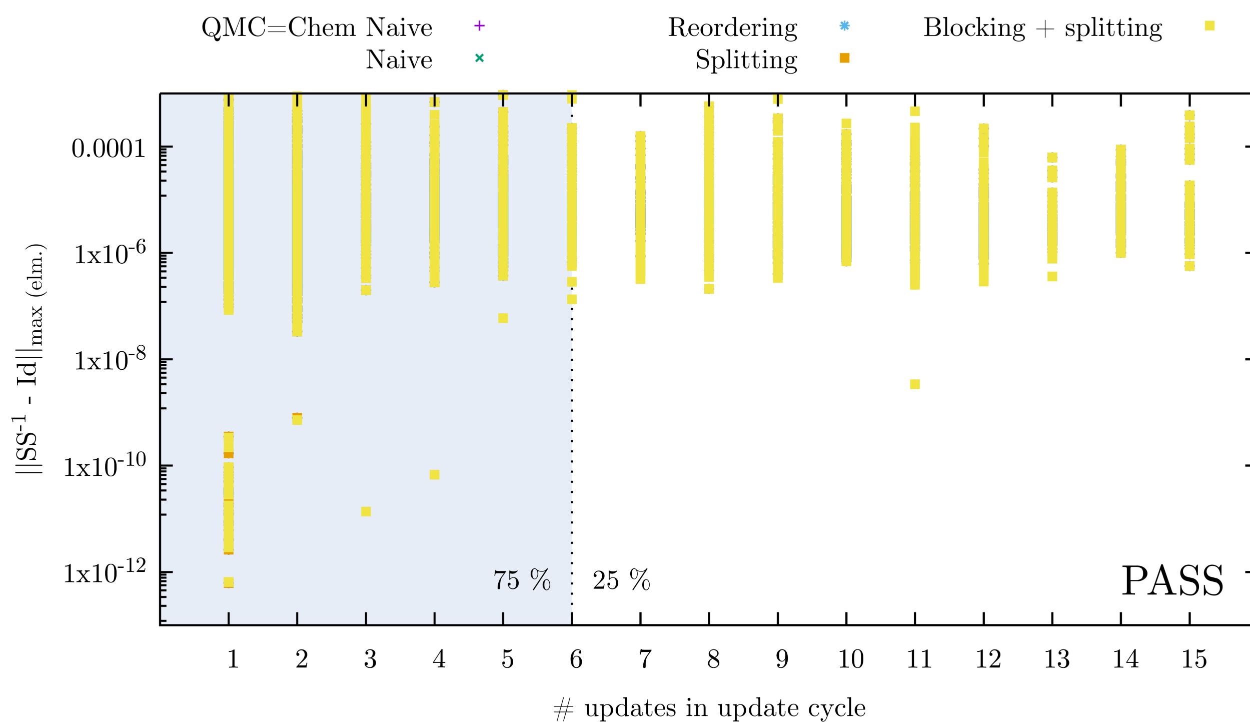

Figure 8: The influence of the number of rank-1 updates to numerical accuracy for various kenrnes.

Naïve kernel is by far the worst as rank-k, with $k > 1$, are very common. It is surprising however that for the rank-1 update case the Naïve kernel seems to do ever so slightly worse than the Splitting kernel, but maybe we are actually looking at the machine noise level; they are the same upto 4 decimal digits. For the rank-2 update case WB2 is slightly worse than Splitting, Reordering and Blocking because the Woodbury kernels seem cause slightly larger loss of precision than the SM-only kernels. The Blocking kernel behaves the same as the Splitting-kernel because it retries the failed cases of WB2 (because the denominator is too small) by sending them to the Splitting kernel and then passes. For WB3 we see the same behaviour for the rank-3 update cases. The failing blocks are retried using the Splitting kernel. Overall the Blocking and Splitting kernels behave numerically equivalent and are the best compared to the others. The Blocking kernel is a few more failing cases because WB2 and WB3 are slightly less accurate, but as we will see later this comes with a slight increase in speed.

Results for the performance

Before we jump to the final performance results there are two aspects that are interesting on their own right. One of them is the performance per rank-1 update, which as will be shown varies little. Then we will consider performance stratification.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

# upds (222008 cycles)	Kernel: Naïve	Splitting	Reordering	WB2	WB3	Blocking
all (222008 cycles)	48.780	.831	.932	–	–	.831
1 (23533 cycles)	.939	.931	.939	–	–	.931
2 (29388 cycles)	5.962	.759	.769	.783	–	.759
3 (26239 cycles)	54.545	.808	.873	–	.842	.808
6 (19442 cycles)	73.835	.741	.905	–	–	.746

Table 2: Fail rates for Benzene with 15784 α-determinants, 222008 determinants in total, $\beta = 1 \times 10^{-3}$ and $\tau = 1 \times 10^{-3}$. The fail rates are shown for five relevant number of updates.

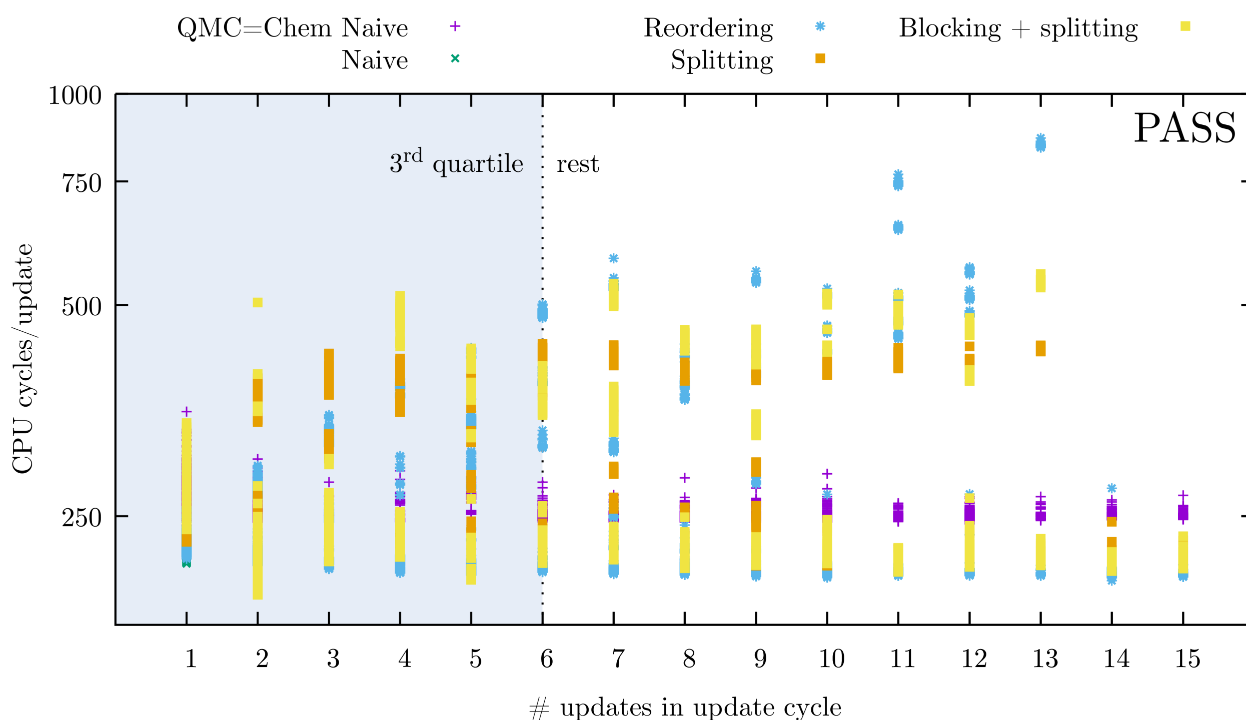

Figure 9: Performance in CPU-cycles per rank-1 update for various kernels.

Performance per rank-1 update Fig. 9 shows the number of CPU-cycles per rank-1 update versus the number of rank-1 updates for various kernels. Focussing our attention on the 3$^{\text{rd}}$ there are two main branches. One is centred around 250 CPU-cycles/rank-1 update. For one rank-1 update the spread is due to variability of the machine itself. For the cases of more than one rank-1 update this is mainly due to the number of splits for the Splitting kernel, or the number of failed blocks in the Blocking kernel.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

Looking at the left two panels of Fig. 10, for the case of Benzene, 329 determinants the maximum number of splits is always one less than the number of updates, in case of the Splitting kernel. The Blocking kernel seems to get away with less splits in general, except in the case of two rank-1 updates. On the right two panels of Fig. 10 we plotted the average number of CPU cycles per rank-1 update as a function of the number of splits. In the PASSING cases the number of CPU cycles per update increase linearly as the number of splits increases, up until 6 splits where it plateaus. For the FAILING cases there are never more than 3 splits, but most of these occurrences are clustered around 0–2 splits and take an average of about 300 CPU cycles per update.

Performance stratification When we plot the element-wise max-norm of the residual matrix ρ against the number of CPU cycles for all kernels, we get the picture that is shown in Figure 11. Each coloured point on this plane corresponds to exactly one update chain computed for the kernel corresponding to that colour. Where this point is located on the plain, together with all the others of the same colour, eventually determines the numerical accuracy and performance of the kernel. When looking at Figure 11 carefully we can see clusters of vertical stripes, except for the LAPACK kernel. Each of these clusters corresponds to a specific set of circumstances. E.g. the green cluster of stripes in the top-left corner are all above the tolerance threshold τ and are failing update-chains. The vertical stripes are formed by how many updates were applied successfully by the "Naïve kernel" before a break-down occurred.

The cluster in the bottom-left between 200 and 300 CPU cycles/update are the update chains that correspond the first ones after a full inversion with LAPACK and have only one rank-1 update. So they did not have the time to accumulate any numerical noise, hence their relatively small residual. Then there is the big cluster in the middle covering most remaining cases. The light blue stripes are formed by the recursions of the "Reordering" kernel. Some update chains are harder than others so sometimes it takes more than a few recursive calls to find a successful order. This accounts for the rather wider range of required CPU cycles/update than that of the "Splitting" and "Blocking+splitting" kernels. To look a bit closer at the big cluster in the middle we selected only the update-chains containing nine rank-1 updates and only plotted the ones that passed with the ''Splitting" kernel. This is shown in the upper panel of Figure 12. In this case the vertical stripes are formed by splits, where the spacing between the stripes correspond to how many times the kernel had to split an update before it could be successfully applied. Not every possible number of splits occurs, as can be seen in the right panel of Figure 12. Here we separated the points by the number of splits. In the case of nine rank-1 updates, only 0, 1, 3, 7 and 8 splits occur, hence difference in the spacing of the stripes.

3.8 Classification of all the kernels

To have a good idea of the numerical accuracy and performance of each of the kernels we have included Figure 13. For each kernel we averaged the results for the numerical accuracy along the vertical axis, the element-wise max-norm of ρ, and took the median of the number of CPU cycles per rank-1 update along the horizontal axes. This results in an easy to read overview for each kernel.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

Ideally the best kernel resides in the lower-left corner of the square and the worst possible one in the upper-right corner.

What is immediately apparent when looking at the left panel, is that even though the inverses computed directly with Intel's MKL are the most accurate (lowest max-norm of ρ), it is also the most costly. Similarly, even though the Naïve kernel is the fastest, it is also the least accurate in cases where there are more than one rank-1 update.

To have a better resolution in the part where the remaining kernels are clustered together the MKL kernel has been removed in the second panel. In the lower-left corner we can see the two kernels Splitting and Blocking+splitting (pink square and red dot). Their performance is similar to the Reordering kernel (green square) but have a better numerical accuracy of at least one order of magnitude. This is good news because it should help with the issues raised in section 3.3.

For some reason the Woodbury $K \times K$ kernel performs relatively bad in terms of accuracy and performance.

3.9 Final recommendations

Summarising the previous results we can conclude with the following recommendations

Single rank-1 updates When there are only single rank-1 updates we don't have to worry about the relatively low numerical accuracy of the Naïve kernel because this is only due to the possible singular behaviour induced by iterative process in multi rank-1 updates. In this case the **Naïve** will give you the best perforce overall.

All other cases In all other cases the Blocking+Splitting kernel is the best option. It has the same numerical accuracy as the Splitting kernel but with the benefit of being slightly faster due to its higher arithmetic intensity.

4 Impact of Mixed-precision in CHAMP

In this numerical exploration of CHAMP [16] we used the Delta-Debug functionality of the numerical analysis tool Verificarlo to analyse the sensitivity of the total energy (TE) and inter-nuclear forces (INF) to numerical noise. On the tested datasets we conclude that for the TE we can reduce most of the code to single precision (SP) floating-point (FP) numbers without sacrificing numerical accuracy. The INFs are more sensitive and only a few functions can be reduced to SP. This might negate any speedup gained from converting said functions to SP.

4.1 Verificarlo and the VPREC backend

Verificarlo (VFC) [17] is a tool for debugging and assessing FP precision and reproducibility. It is an extension to the LLVM compiler that can use various arithmetic backends. In this exploration we

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

used the VPREC backend libinterflop_vprec.so. It supports all the major languages including C, C++, and Fortran. Unlike source-to-source approaches, it captures the influence of compiler optimisations on the numerical accuracy.

The VPREC backend simulates any FP formats that can fit into the IEEE-754 double precision format [7]. The backend allows modifying the bit length of the exponent (range) and the pseudo-mantissa (precision). Here we only used VPREC to modify the pseudo-mantissa.

4.2 Delta-Debug and VFC-DD

Delta-Debug (DD) method is a generic bug-reduction method [18] that allows to efficiently find minimal sets of conditions that trigger a bug. In the context of Verificarlo Delta-Debug (VFC-DD), which is an implementation of the DD method that is guided by VFC, the 'conditions' are the dependencies of the quantities-of-interest's numerical precision and a 'bug' means: whenever the numerical accuracy of these quantities-of-interest, that depend on these conditions, drop below a set amount of significant digits. When this happens the program is said to "fail" and otherwise it is said to "pass".

Once VFC-DD knows what 'passing' and 'failing' means it uses a systematic way to inject a user-configurable amount of noise in some FP operations during program execution, but not in others. By using this method VFC-DD will output one or more 'minimal sets' called ddmin{X}, where X is the number of the minimal set. Each of these minimal sets contain a minimal configuration of conditions that trigger a bug. It is then the complement of the union of these ddmin{X} sets that gives a list of quantities that can be safely moved to a lower precision without sacrificing the numerical accuracy of the quantities-of-interest.

With this list in hand we can then set out to reduce the precision in some "hot" parts of the code in the hope to gain a speed-up without sacrificing numerical accuracy. If you want to know how the method works from a conceptual point of view, I encourage you to read Chapter 4 in reference [18].

4.3 Methodology

In order for Verificarlo and VFC-DD to work with CHAMP, Verificarlo needs Flang as its frontend to LLVM to compile CHAMP. To compile CHAMP with Flang a few minor changes needed to be made. The version of CHAMP that we used is based on commit:

```
https://github.com/
        filippi-claudia/champ/commit/
        f84d4e2037c914b38d82dba63abafd1f860a3c51
```

No changes have been made that change the behaviour of CHAMP. The changes made are to guarantee successful compilation with Verificarlo/Flang. To enable VFC-DD functionality in CHAMP the compiler flag --ddebug needs to be passed:

```
verificarlo-{c/c++/f} --ddebug -c <input.{c/cpp/f/f90}>
```

After that, CHAMP/vmc.mov1 is run on a configuration of Butadiene

```
/path/to/champ/bin/vmc.mov1 -i butadiene.inp -o butadiene.out
```

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

During the measurements we have chosen to monitor the number of significant digits of the total energy and inter-nuclear forces. They are monitored by extracting them from the output file `butadiene.out` generated by CHAMP. For these experiments we monitored only `FORCE 1` and ignored the forces of the other three nuclei since they are computed using the same code path.

The VFC-DD run is executed by using the Python script `vfc_ddebug` that needs two arguments. The first argument is a script that takes care of running the executable that needs to be analysed, commonly called `ddRun`. It can be any kind of script, Shell, Python, Perl, etc.

The second argument is a script that compares the values of a specific run with injected noise against the values of a reference run done at the beginning. This script is assumed to fail when a certain criteria is not met, e.g. the number of significant digits drops below 5. It is commonly named `ddCmp` and can be written in any kind of language as long is it is able to return exit code '0' back to the shell in case of success and anything else in case of fail. A minimal example that applies VFC-DD to Archimedes' method for computing π with `ddRun` and `ddCmp` scripts can be found here: https://github.com/verificarlo/verificarlo_tutorial/tree/master/verificarlo-tutorial/archimedes.

Once the VFC-DD is started `ddRun` runs CHAMP. The first run is done with the IEEE backend which produces the reference values. After that a second run is done with the VPREC backend where the mantissa is cut at a preset value of 24 bits. The python script `ddCmp` is then run to compare the two values and calculate the number of significant binary digits with the following formula

$$s = -\log_2 \left| \frac{x_{\text{VPREC}} - x_{\text{IEEE}}}{x_{\text{IEEE}}} \right| \tag{29}$$

where x_{VPREC} is the monitored value obtained from a run with the VPREC backend and x_{IEEE} is the reference value obtained from the first run with the IEEE backend.

4.4 Mixed precision experiments on CHAMP

Arithmetic operations in SP takes half the amount of time of the same operation in DP, so the expected speed-up is 2×. But in reality it is a bit more complicated. Moving to SP will lead to an overall reduction in arithmetic operations and in case of vectorised computations it can potentially fit more elements in a vector if there are less bits per element. Moreover, there could be a gain due to a reduction in memory size that causes better cache occupation, therefore reducing communication over cache lanes.

These effects can sometimes compound and lead to super-linear behaviour in the speed-up but, this is very rare. In memory- or communication-bound codes, most of the gains can come from using less memory. In most cases the speed-up will be somewhere between 1 and 2.

So, if DP is used everywhere where it is not needed, then the code could be slower than it needs to be. The obvious thing to do would be to switch to SP and have a potential speedup.

Total energy We found that as far as the TE is concerned, see Table 3, all the code, except subroutine `splfit` in `splfit.f` can be switched to SP without sacrificing numerical accuracy.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

Function name	Required precision (VPREC)
`splfit`	Double
`ALL OTHERS`	Single

Table 3: CHAMP/VMC Butadiene CIPSI. Investigated precision on TOTAL ENERGY for all functions.

4.5 Inter-nuclear forces

The INFs, see Table 4 are slightly more sensitive to numerical noise. Most functions, containing about 81% of the selected lines, need to stay in DP. But other functions, containing the remaining 19% of selected lines, can be changed to SP. The number of functions containing lines that can be moved to SP is still rather long and would require a significant amount of manual labor to move to SP and test, and this for a yet unknown amount of gain in speed. We have therefore cross-referenced this list of functions with a list of functions obtained with a hotspot analysis using MAQAO ONE View [19]. From this list we selected the functions that make-up more than 2% of the execution time. They are listed in Table 4. From the functions that take $\geq 5\%$ of the execution time we list in the last column what precision they require.

For example, we manually switched the subroutine `orbitals_quad` to SP and checked that the numerical accuracy was not affected by the change. This is indeed the case but since we cannot change all the code to SP one needs to make sure that the arguments that are passed are precision-matched before and after the call. This will an additional performance cost that needs to be balanced against the potential gain of switching to SP.

In the list of hotspots in Table 4 also appear the functions `__powr8i4` and `__libm_log_l9`. In Intel's Short Vector Math Library (SVML) there exist three versions of these functions; a low precision version, called Enhanced Performance (EP), a medium precision version and a high accuracy version, called High Accuracy (HA). To see if there is something to be gained we also experimented with the three different version of these functions in CHAMP. To do this, additional Fortran compiler flags need to be passed to the Intel compiler:

```
-fimf-precision={low,medium,high}:pow,log
-fimf-use-svml
```

In the first flag the user can select the desired precision for each arithmetic function. In this case only the precision of the `pow` and `log` functions are modified. The other ones remain at the default precision. If non of these flags are passed the default precision is used for all the SVML functions, which is set to `medium`.

In Table 5 the results are shown for CHAMP running in 'Standard' mode, that is, with no flags passed, and the three different precision modes of SVML in increasing precision. For each precision CHAMP is run $25\times$ to get a decent mean and STD for the execution times.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

	Time spent (%)		
Function name	500 dets	15k dets	Required precision (VPREC)
orbitals	22.02	5.71	Double
nonloc	11.7	3.15	
> orbitals_quad:395	7		Single
orbitalse	5.56	3.36	Not called in VFC-DD
optjas_deloc	4.66	16.45	Singel
splfit	4		Double
multideterminante_grad	3.62	2.48	
multideterminante	3.59	14.86	Double
multideterminant_hpsi	3.56	3.49	
n0_inc	2.6		
basis_fns_vgl	2.48		
basis_fnse_v	2.43		
optorb_compute	2.08		
compute_ymat		12.8	Double
detsav		4.66	Not called in VFC-DD
__powr8i4	7.16	1.88	
__libm_log_l9	2.43	0.58	

Table 4: CHAMP/VMC Butadiene CIPSI. Investigated precision on FORCES for functions of runtime $>= 5\%$ of total run-time.

The results seem to suggest that the Standard mode of CHAMP has most in common with the EP version, both in execution time and STD on the TE. We don't understand this yet. The smallest error on the TE is obtained with the HA version, while the execution time is not significantly larger than that of the 'Low' and 'Medium' ones.

4.6 Recommendations

Single Precision only mode Since the TE is for the largest part completely insensitive to the injected numerical noise, the idea is to use, e.g. preprocessor directives give the ability to choose

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

Precision	mean run-time (s)	STD run-time (s)	Total energy (Hartree)
Standard	18.761	0.12475	$-26.2253208 \pm 0.0089076$
Low (EP)	18.861	0.12303	$-26.2255374 \pm 0.0083120$
Medium	18.966	0.10977	$-26.2344844 \pm 0.0097661$
High (HA)	19.199	0.11387	$-26.2249791 \pm 0.0076948$

Table 5: Run-times for VMC.MOV1 for 3 different precision levels of Intel Short Vector Math Library (SVML) operations `pow` and `log`.

between a Low Accuracy mode if one is only interested in the TE, and a High Accuracy mode if one is also interested in computing inter-nuclear forces. This can be done by switching all the variables in `VMC.mov1` to SP, with the exception of subroutine `splfit`. In the High Accuracy mode all the variables can be kept in DP with the exception of maybe the subroutines `orbitals_quad` and `optjas_deloc`.

Try on different datasets All this work has been done on Butadiene. We do not know if the numerical behaviour is much different when other materials are investigated. Considering the fact for example that the INFs sometimes experience violent fluctuations whenever the reference determinant gets too close to zero, depending on the input parameters, it would be prudent to see how CHAMP/VMC behaves for other materials.

Move to Intel SVML High Accuracy for `pow` and `log` Table 5 suggests that moving the arithmetic functions from medium to high accuracy comes with a very small cost in performance, only $1.02\times$ slower. An added benefit might by that during the Monte-Carlo process, because of the higher accuracy, less iterations might be needed to reach the same level of convergence on the energy. We don't know if this is indeed the case, or how to check this. It could be interesting to see how to check this. Additionally, moving to a larger number of determinants seems to diminish the time spent in pow and log, so the move to HA should be come even less costly.

Optimisations on forces probably marginal Due to the possible overhead introduced by the additional type conversions at the boundaries of function calls there might be no net speed-up. We tried moving subroutine `orbitals_quad` to SP and did the type conversion before and after the functions call. We noticed no appreciable speedup. We therefore advise not to move the functions in Table 4 that can be moved to SP (e.g. `orbitals_quad` and `optjas_deloc`) to SP as it will cost a significant amount of manual labour with a possible marginal speedup.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

5 Conclusion

Verificarlo CI reuses the core backends of Verificarlo and therefore supplements the other existing Verificarlo post-processing tools. To the best of our knowledge, it is the first tool that automates numerical accuracy tests within a continuous integration workflow: it grants users the ability to define such tests, and provides an easy way to visualize results throughout the development process of a code. Verificarlo CI it is completely domain agnostic and could potentially be used on any code using floating point in a variety of domain. Better integration of numerical concern in CI process make the development processes safer and debugging easier, thus saving code developer precious time to focus on their area of expertise. This work has been released within Verificarlo open-source project which is available at https://github.com/verificarlo/verificarlo. A tutorial focusing on Verificarlo CI is available at at https://github.com/verificarlo/vfc_ci_tutorial.

In future works, it would be interesting to validate these results on others datasets extending to other materials than Butadiene. In the same spirit, we would like to numerically validate the proposed improvements on other computing architectures such as ARM. On September 2023, we have released a new version of Verificarlo fully supporting ARM64 architectures. Section 3 focused on the numerical and performance optimization of the SMWB kernel, which is an important hotspot in multiple QMC codes. In future works, it would be interesting to study other hotspots, such as Ewald summations (important for example in TurboRVB).

All the SMWB computational kernels that are presented in this report are included in the TREX Centre of Excellence Quantum Monte Carlo Kernel Library [20] (TREX-CoE/QMCkl). The repository where the QMCkl code and documentation is stored can be found on https://github.com/TREX-CoE/qmckl.

The kernels themselves as well as the code, documentation and regression testes can be found in an 'Org-mode' formatted file can be found the org/ subdirectory at https://github.com/TREX-CoE/qmckl/blob/master/org/qmckl_sherman_morrison_woodbury.org.

The HPC versions of these kernels can be found at https://github.com/TREX-CoE/qmckl/tree/master/org/hpc.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

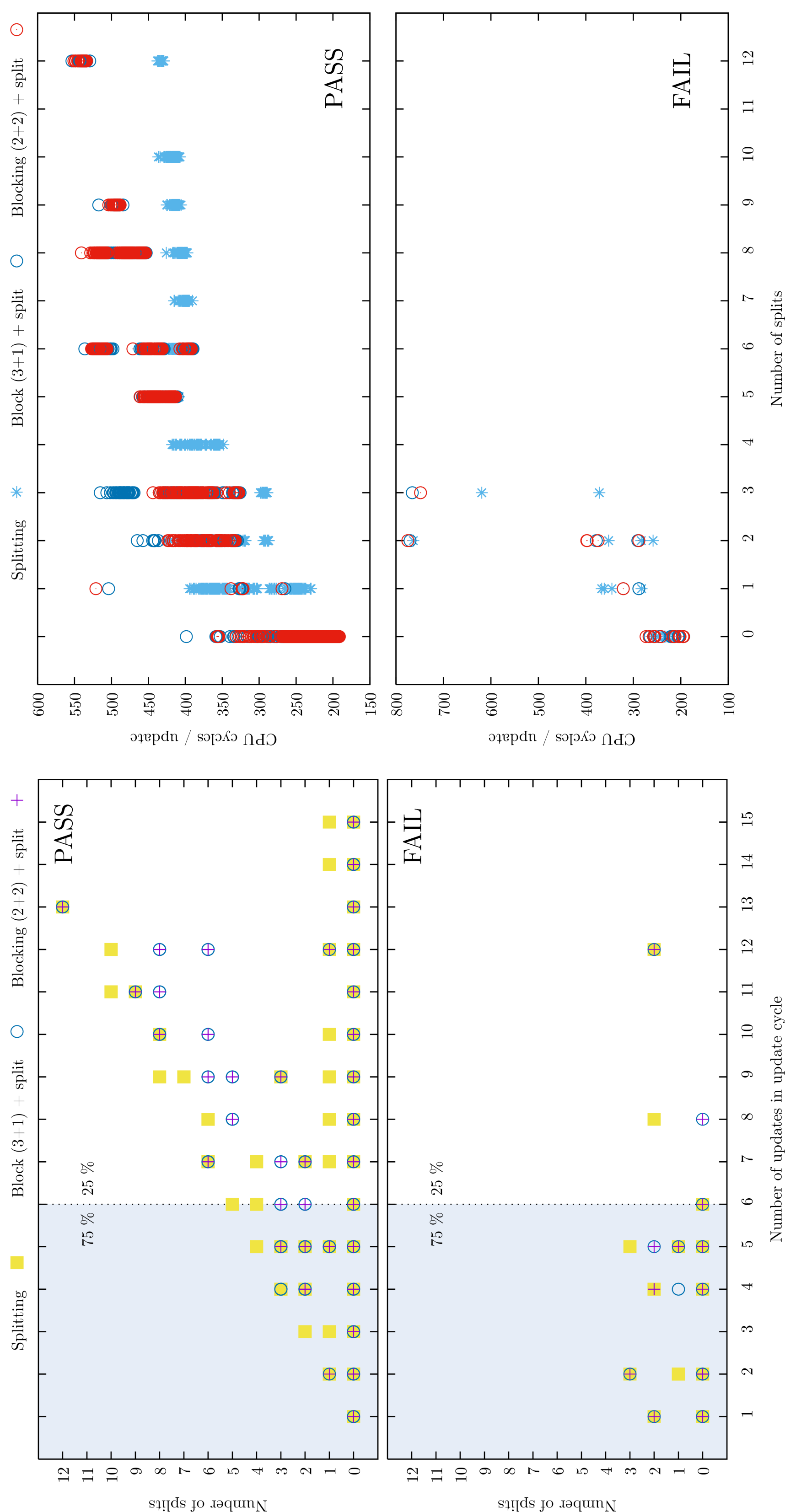

Figure 10: Distribution of the occurrence frequency for each number of rank-1 updates that can occur. The blue area represents the 3rd quartile of the distribution. For the 329-determinants case, 75% of the time updates consist of 1–6 rank-1 updates. For the 15784-determinants case updates consist of 1–8 rank-1 updates.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

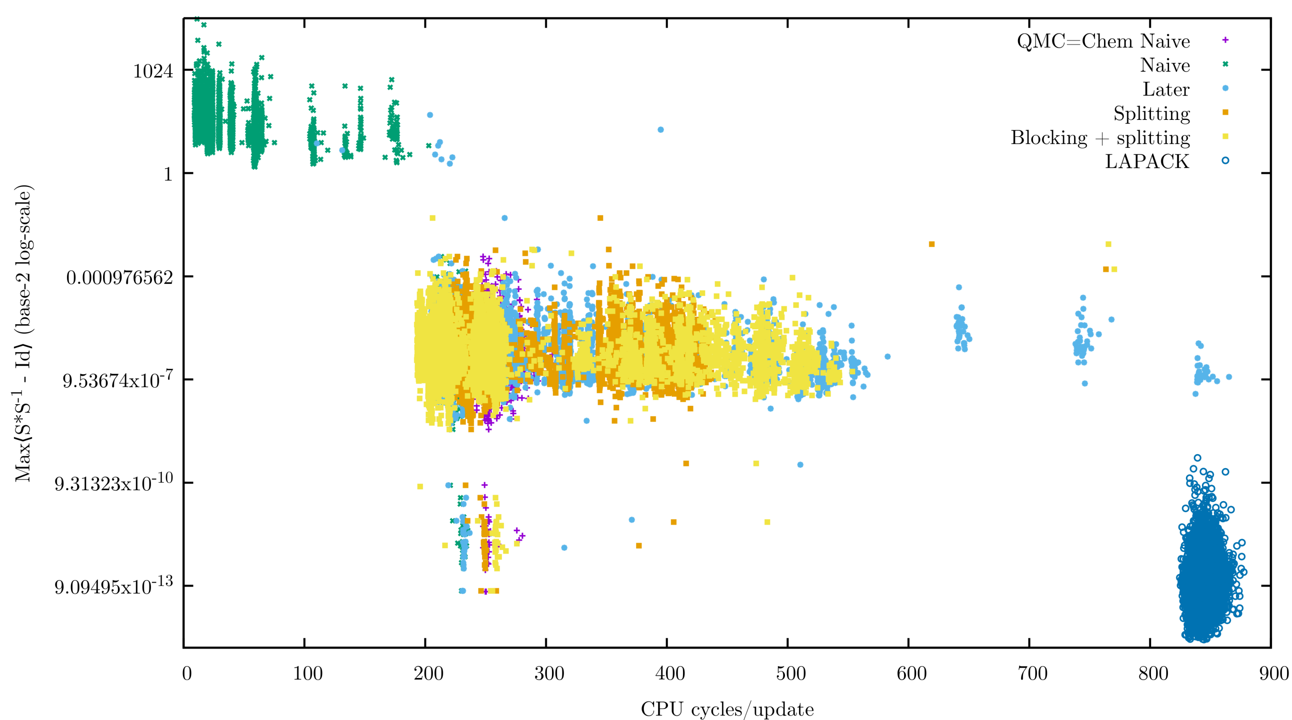

Figure 11: The element-wise max-norm of the residual matrix ρ as a function of the number of CPU cycles (averaged) for all tested kernels.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

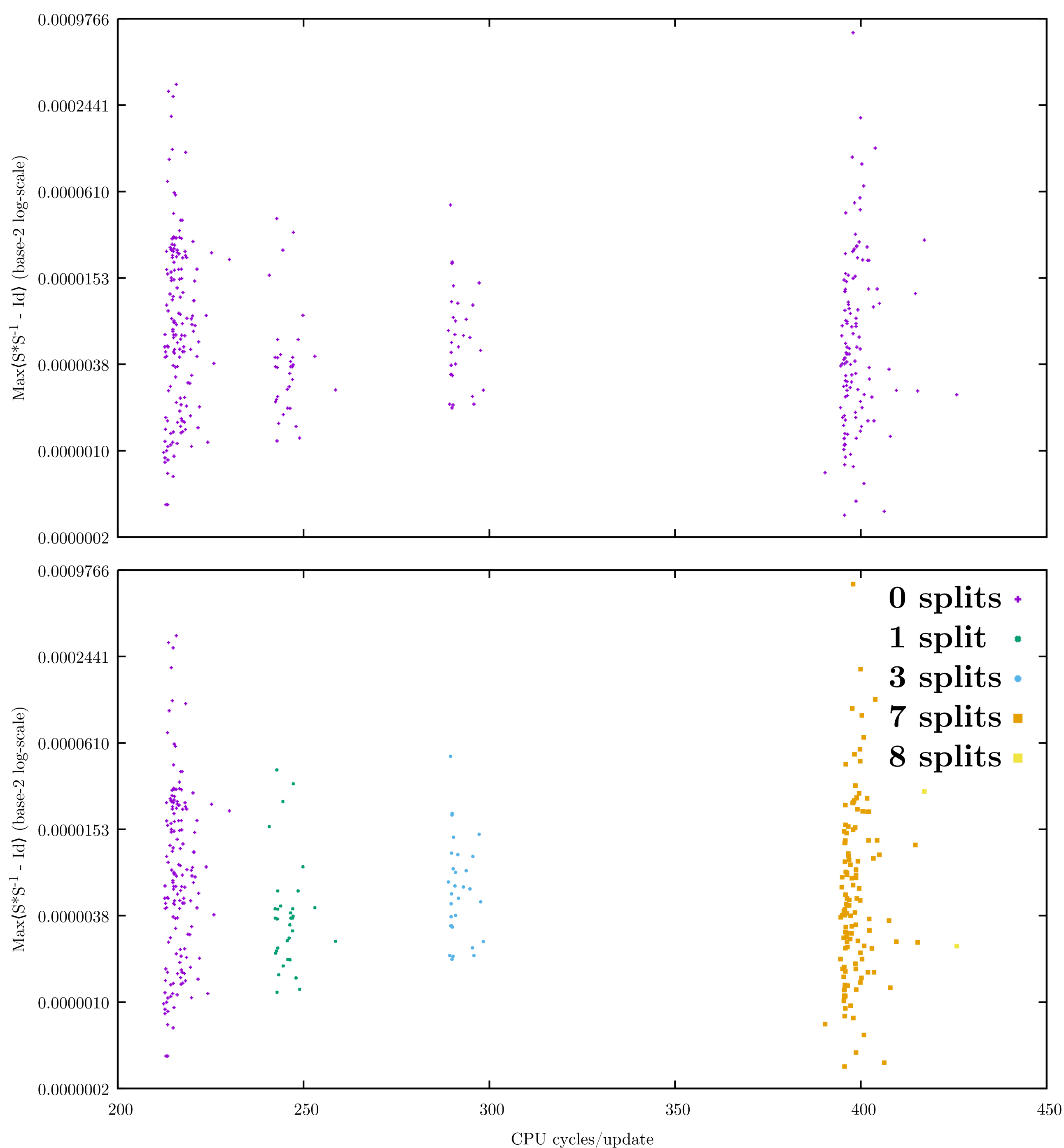

Figure 12: Performance stratification caused by update splitting.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

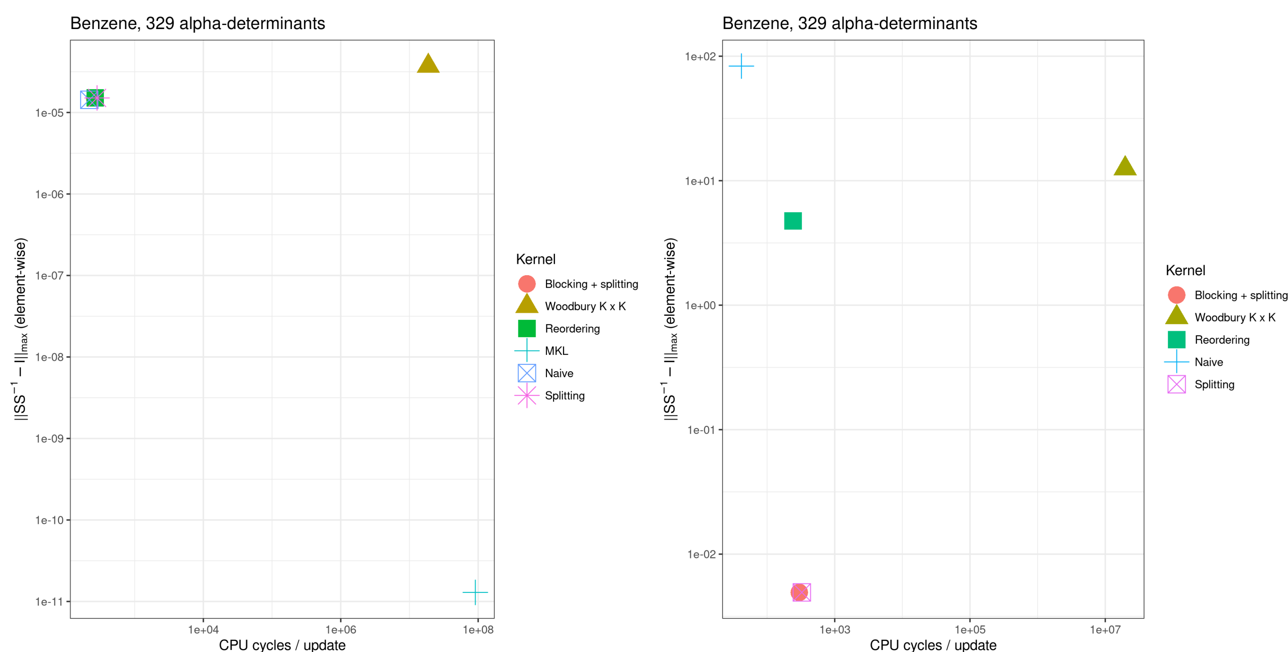

Figure 13: Numerical accuracy versus performance

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

References

[1] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Comput. Surv.*, 1991.

[2] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.

[3] C. D., P. de Oliveira Castro, and E. Petit, "Verificarlo: Checking floating point accuracy through monte carlo arithmetic," in *23nd IEEE Symposium on Computer Arithmetic, ARITH 2016*, 2016.

[4] P. de Oliveira Castro, E. Petit, and Y. C. et al, "Verificarlo v0.9.1, 10.5281/zenodo.7235096," Oct. 2022.

[5] D. S. Parker, "Monte carlo arithmetic: exploiting randomness in floating-point arithmetic," no. CSD-970002, 1997.

[6] D. Sohier, P. de Oliveira Castro, F. Févotte, B. Lathuilière, E. Petit, and O. Jamond, "Confidence intervals for stochastic arithmetic," *https://arxiv.org/abs/1807.09655*, 2021.

[7] Y. Chatelain, E. Petit, P. de Oliveira Castro, G. Lartigue, and D. Defour, "Automatic exploration of reduced floating-point representations in iterative methods," in *Euro-Par 2019 Parallel Processing - 25th International Conference*, 2019.

[8] A. Scemama, "Quantum Monte Carlo for Chemistry," http://qmcchem.ups-tlse.fr/index.php/Quantum_Monte_Carlo_for_Chemistry_@_Toulouse, 2015.

[9] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[10] *Intel Math Kernel Library. Reference Manual.* Intel Corporation, 2009, santa Clara, USA. ISBN 630813-054US. [Online]. Available: https://www.intel.com/content/www/us/en/develop/documentation/onemkl-developer-reference-c/top.html

[11] P. Maponi, "The solution of linear systems by using the sherman–morrison formula," *Linear Algebra and its Applications*, vol. 420, no. 2, pp. 276–294, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0024379506003351

[12] J. T. Slagel, "The sherman morrison iteration," Master's thesis, Virginia Polytechnic Institute and State University, 2015.

[13] P. Maponi, "The solution of linear systems by using the sherman–morrison formula," *Linear algebra and its applications*, vol. 420, 2007.

[14] J. T. Slagel, "The sherman morrison iteration," Master's thesis, Virginia Tech, 2015.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.

[15] F. Coppens, "Extracted QMC=Chem datasets that were used in producing the results in this report," rsync://trex@trex-share.univ-tlse3.fr:uvsq/, 2021.

[16] C. Umrigar and C. Filippi, "Cornell-holland ab-initio materials package (champ)," *The code can be used for finite and extended systems. See https://www.utwente.nl/en/tnw/ccp/research/CHAMP.html*, 2019.

[17] C. Denis, P. de Oliveira Castro, and E. Petit, "Verificarlo: Checking floating point accuracy through monte carlo arithmetic," in *23nd IEEE Symposium on Computer Arithmetic, ARITH 2016, Silicon Valley, CA, USA, July 10-13, 2016*, 2016, pp. 55–62. [Online]. Available: http://dx.doi.org/10.1109/ARITH.2016.31

[18] A. Zeller, "Yesterday, my program worked. today, it does not. why?" *SIGSOFT Softw. Eng. Notes*, vol. 24, no. 6, p. 253–267, oct 1999. [Online]. Available: https://doi.org/10.1145/318774.318946

[19] W. Jalby, C. Valensi, M. Tribalat, K. Camus, Y. Lebras, E. Oseret, and S. Ibnamar, "One view: A fully automatic method for aggregating key performance metrics and providing users with a synthetic view of hpc applications," in *Tools for High Performance Computing 2018 / 2019*, H. Mix, C. Niethammer, H. Zhou, W. E. Nagel, and M. M. Resch, Eds. Cham: Springer International Publishing, 2021, pp. 219–235.

[20] "The QMCkl library for software developers and HPC experts," https://github.com/TREX-CoE/qmckl, 2020.

TREX: Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation program under Grant Agreement No. 952165.