D1.2 – Report on pre-release of open-source human-readable implementation of QMCkl

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)

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	D1.2
Deliverable title	D1.2 – Report on pre-release of open-source human-readable implementation of QMCkl, as seen in GA
Due Date	M13 – 31-10-2021 (from GA)
Actual Submission Date	29-10-2021
Work Package	WP1 – Standard API for QMC kernels and implementation
Lead Author (Org)	Anthony Scemama (Centre National de la Recherche Scientifique (CNRS))
Contributing Author(s) (Org)	Vijay Gopal Chilkuri (CNRS), Aurélien Delval (Université de Versailles Saint-Quentin-en-yvelines (UVSQ)), Pablo Oliveira (UVSQ)
Reviewers (Org)	Fabio Affinito (Consorzio Interuniversitario (CINECA)), Ivan Štich (Slovak Academy of Sciences (SAV)), Jan Beerens (Universiteit Twente (UT)
Version	1.0
Dissemination level	PU
Nature	Report
Draft / final	Final
No. of pages including cover	26

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	29-10-2021	Anthony Scemama (CNRS)	First Official Release

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.

Abbreviations

AO	Atomic Orbital
API	Application Programming Interface
BLAS	Basic Linear Algebra Subprograms
CINECA	Consorzio Interuniversitario
CNRS	Centre National de la Recherche Scientifique
CoE	Center of Excellence
DFT	Density Functional Theory
DMC	Diffusion Monte Carlo
ECP	Effective Core Potential
GPU	Graphical Processing Unit
HPC	High Perfomance Computing
MKL	Math Kernel Library
MO	Molecular Orbital
QMCkl	Quantum Monte Carlo kernel library
QMC	Quantum Monte Carlo
SAV	Slovak Academy of Sciences
TREX	Targeting REal chemical accuracy at the eXascale
TREXIO	TREX Input/Output
UT	Universiteit Twente
UVSQ	Université de Versailles Saint-Quentin-en-yvelines
WP	Work Package

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

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.

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.

List of Figures

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.

1 Introduction

There are three different Quantum Monte Carlo (QMC) codes in the Targeting REal chemical accuracy at the eXascale (TREX) Center of Excellence (CoE), TurboRVB, Champ, and QMC=Chem, each with its own strengths and weaknesses. Instead of optimizing the three codes independently for exascale architectures, or re-writing a new monolithic code, our strategy is instead to design a new library, Quantum Monte Carlo kernel library (QMCkl), containing the state-of-the-art expertise in the implementation of specific kernels present in each of the three codes. The functions available in this library will allow all the codes to benefit from the optimal implementation of the major kernels of QMC. The three main objectives driving the development of QMCkl are *performance*, *productivity* and *portability*.

Performance

The ultimate goal of QMCkl is to provide a high-performance implementation of the main computational kernels involved in QMC methods, ready to be adapted on exascale machines. In Work Package (WP)1, we focus on the definition of the Application Programming Interface (API), the tests, and on a *pedagogical* presentation of the algorithms. The high-performance aspects are delegated to WP3, but there is nevertheless a strong interaction between these two WPs because the design of the data structures and the API are guided by the possibility to enable a high-performance implementation of the computational kernels. For instance, the data structures should be such that the memory access patterns are efficient both on CPUs and accelerators, and the library should not be too restrictive on how the memory is allocated to give enough freedom to the developers of the High Perfomance Computing (HPC) variants to allocate memory on the CPU or Graphical Processing Unit (GPU), pinned or not, etc.

The main objective of this pedagogical implementation of the library is to provide a reference implementation of the kernels such that HPC experts involved in WP3 will be able to use it to rewrite optimized versions of the functions described in this library, using the same API.

Portability

QMCkl should take advantage of exascale machines. In terms of hardware, many different architectural designs are proposed for exascale supercomputers. Some are CPU-based, others are GPU-based, multiple vendors propose different CPUs (Intel, AMD, ARM, ...) or GPUs (Intel, NVidia, AMD, ...). In this context, we have to propose a library that will take advantage of any of these combinations so we can't rely on a vendor-specific software stack like Intel's Math Kernel Library (MKL) or Nvidia's Cuda language. Instead, we will rely on free software and open standards such as OpenMP, but the BSD licence we have adopted also gives the freedom to the community to modify QMCkl and propose vendor-specific implementations of QMCkl.

For maximum portability, we have chosen to propose an API compatible with the C language. In this way, we guarantee that QMCkl will be usable in *all* the QMC codes used by the QMC community, although these codes are written in different programming languages (C, C++, Python, Fortran, etc.).

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.

Productivity

QMC methods are still under heavy development. Therefore, scientists need to be able to understand the code to modify the algorithms and propose new methods. It is not realistic to believe that any physicist/chemist will be able to read and develop a highly optimized C++ code tuned by HPC experts. Similarly, it is not realistic either to believe that a code written by a random physicist/chemist will be easily ported to GPU by a HPC expert without requiring a deep restructuring of the code. If a compromise is chosen between these two extremes, it is likely that it will converge to a code difficult to maintain by the physicist and difficult to optimize by the HPC expert. Therefore our choice is to develop at least two libraries with the same API: first a *pedagogical* implementation (the objective of the current WP) designed for physicists/chemists, and then multiple high-performance implementations of this library.

2 Development environment

In the first months, a large effort was put in preparing the foundations of the library: the coding standards and rules, GNU Autotools configuration scripts, Makefiles, scripts to generate automatically the code and the documentation, unit testing, continuous integration, etc. It is important not to neglect the importance of all these developments which are not directly visible from outside of the project. Having spent a large effort of preparation pays a lot in the long term since is can dramatically enhance the productivity of the contributors of the library. We present in this section the major efforts that lead us to being highly productive in implementing new kernels.

Literate programming

As the focus of this pedagogical implementation of QMCkl is documentation, we use literate programming using `org-mode` files.[1, 2] Hence, the code and the documentation are produced with the same source files, allowing the programmers to write LaTeX formulas and use tables or figures together with the corresponding code in the same files. This method has proven to be particularly efficient in maintaining the documentation consistent with the source code, and literate programming is now becoming a very popular technique thanks to the development of Jupyter notebooks. Here, we have chosen `org-mode` as it allows the simultaneous usage of multiple languages in the same notebook, and also doesn't impose to use a web browser for editing.

Control of Numerical accuracy

An original feature of QMCkl is that the numerical precision of the implemented kernels will be monitored during the whole development process of the library. With this aim in mind, the UVSQ team has proposed an internship to a master's student, Aurélien Delval, to build a continuous integration system which, instead of monitoring if tests pass or fail, keeps track of the numerical accuracy of the implemented kernels of QMCkl.

This tools relies on Verificarlo,[3] a software developed by TREX members that implements Monte Carlo arithmetic at the binary level via the LLVM compiler. The continuous integration tool Verificarlo-

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.

CI has been integrated into GitHub actions, such that by inserting probes in the tests provided with QMCKL the numerical accuracy can be monitored each time a new version of the code is pushed on the server.

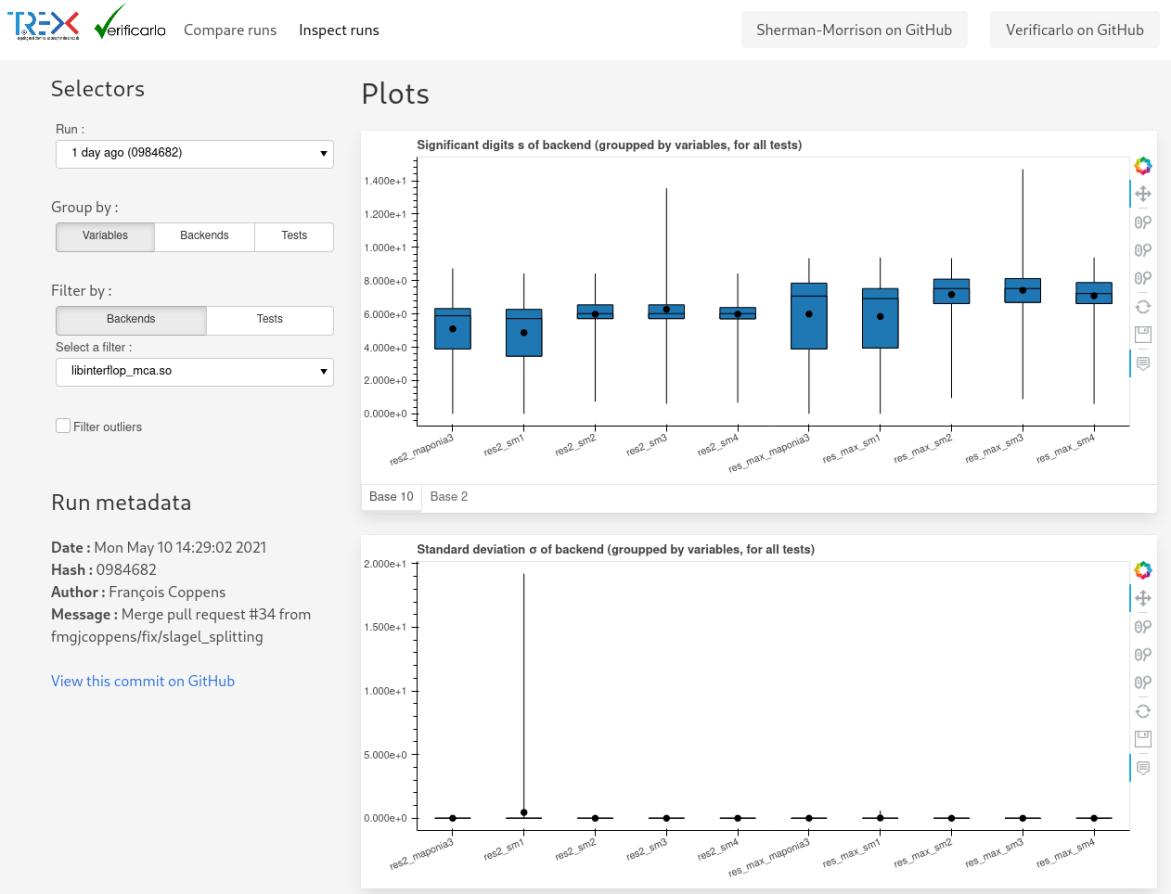


Figure 1: Verificarlo-CI: graphical display of the precision obtained with multiple kernels.

A nice graphical interface is proposed to compare the accuracy obtained with different versions, or to display the fluctuation of the results obtained by different kernels (Fig. 1). A tutorial about how to use Verificarlo-CI is presented in appendix.

Introduction of TREXIO in QMCkl

The input of a QMC calculation is a wave function, which contains usually a large number of parameters of different nature: nuclear coordinates, atomic basis set parameters, pseudo-potential parameters, molecular orbital coefficients, Slater determinant expansion coefficients, etc. Hence, initializing QMCKL can quickly become cumbersome to the users of the library.

TREX Input/Output (TREXIO) is the file format and input/output library developed in WP2. It allows to store all the information relative to a wave function in a file, and provides functions to retrieve easily all these parameters.

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.

To simplify the initialization procedure of the library, we have introduced the possibility to load, with a single function call, all the wave function parameters contained in a TREXIO file. This simple change is extremely beneficial to the user experience.

3 Kernels implemented in QMCkl

After a large effort setting up a productive environment, in the last few months we focused on implementing and documenting kernels. The documentation of the current status of the library is available at https://trex-coe.github.io/qmckl, and the source code is available on the GitHub repository at https://github.com/trex-coe/qmckl. Our objective for this milestone was to provide a library that is capable of computing the local energy (the central quantity in QMC) for any arbitrary system described by a Slater determinant using a Gaussian atomic basis set:

$$E_{\text{loc}}\left(\mathbf{r}_1^{\uparrow},\ldots,\mathbf{r}_{\mathbf{M}}^{\uparrow},\mathbf{r}_{\mathbf{M+1}}^{\downarrow},\ldots,\mathbf{r}_{\mathbf{N}}^{\downarrow}\right)=\frac{\hat{T}\Psi\left(\mathbf{r}_1^{\uparrow},\ldots,\mathbf{r}_{\mathbf{M}}^{\uparrow},\mathbf{r}_{\mathbf{M+1}}^{\downarrow},\ldots,\mathbf{r}_{\mathbf{N}}^{\downarrow}\right)}{\Psi\left(\mathbf{r}_1^{\uparrow},\ldots,\mathbf{r}_{\mathbf{M}}^{\uparrow},\mathbf{r}_{\mathbf{M+1}}^{\downarrow},\ldots,\mathbf{r}_{\mathbf{N}}^{\downarrow}\right)}+V\left(\mathbf{r}_1^{\uparrow},\ldots,\mathbf{r}_{\mathbf{M}}^{\uparrow},\mathbf{r}_{\mathbf{M+1}}^{\downarrow},\ldots,\mathbf{r}_{\mathbf{N}}^{\downarrow}\right)$$

with

$$\Psi\left(\mathbf{r}_1^{\uparrow},\ldots,\mathbf{r}_{\mathbf{M}}^{\uparrow},\mathbf{r}_{\mathbf{M+1}}^{\downarrow},\ldots,\mathbf{r}_{\mathbf{N}}^{\downarrow}\right)=\begin{vmatrix}\phi_1\left(\mathbf{r}_1^{\uparrow}\right) & \cdots & \phi_1\left(\mathbf{r}_M^{\uparrow}\right)\\ \vdots & \ddots & \vdots\\ \phi_M\left(\mathbf{r}_1^{\uparrow}\right) & \cdots & \phi_M\left(\mathbf{r}_M^{\uparrow}\right)\end{vmatrix}\begin{vmatrix}\phi_{M+1}\left(\mathbf{r}_{M+1}^{\downarrow}\right) & \cdots & \phi_{M+1}\left(\mathbf{r}_N^{\downarrow}\right)\\ \vdots & \ddots & \vdots\\ \phi_N\left(\mathbf{r}_{M+1}^{\downarrow}\right) & \cdots & \phi_N\left(\mathbf{r}_N^{\downarrow}\right)\end{vmatrix}$$

We have successfully reached this stage, and the implemented kernels are described in this section.

Low-level kernels

One of the main development guidelines is to maximize the use of Basic Linear Algebra Subprograms (BLAS)-like operations in the main kernels. The main BLAS operations are as follows:

- Level 1 - Operations (e.g. scalar product) on two vectors.
- Level 2 - Operations involving Matrix and vector (e.g. Matrix vector product).
- Level 3 - Operations involving two matrices (Matrix times Matrix).

Incorporating these basic operations in developing the main algorithms is key to achieve an optimal implementation of the library. However, the efficiency of these key BLAS functions depend on the type of hardware (i.e. CPU, or accelerator) and require a tailored approach. Therefore, in QMCKL, we have chosen to abstract the calls to such libraries with standard QMCKL functions such as `qmckl_dgemm` and `qmckl_invert` for matrix multiplication and inversion. This has a two-fold advantage of standardizing the API and enabling hardware specific optimizations by HPC experts without breaking user-space.

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.

A second and significant advantage of providing native implementations of such key functions is the *separation of concerns* approach, which enables new developers and scientists to easily setup and install a minimal version of QMCKL without struggling with a large set of dependencies, thus enabling easy access to new users.

High-level kernels

The electron-electron V_{ee} and nucleus-nucleus V_{NN} repulsion contributions to the potential, as well as the electron-nucleus V_{eN} attraction have been implemented. These kernels are relatively straightforward. However, the computation of the kinetic energy $\hat{T}$ is more complex since it requires the evaluation of the wave function at the electron positions and its second derivative. In the following, we give some information about the kernels implemented for the computation of the kinetic energy.

Atomic basis functions

The atomic basis set is defined as a list of shells. Each shell s is centered on a nucleus A, possesses a given angular momentum l and a radial function R_s. The radial function is a linear combination of *primitive* functions that can be of type Slater ($p = 1$) or Gaussian ($p = 2$):

$$R_s(\mathbf{r}) = \mathcal{N}_s |\mathbf{r} - \mathbf{R}_A|^{n_s} \sum_{k=1}^{N_{\text{prim}}} a_{ks}\, f_{ks} \exp\left(-\gamma_{ks} |\mathbf{r} - \mathbf{R}_A|^p\right).$$

In the case of Gaussian functions, n_s is always zero. The normalization factor $\mathcal{N}_s$ ensures that all the functions of the shell are normalized (integrate to unity). Usually, basis sets are given a combination of normalized primitives, so the normalization coefficients of the primitives, f_{ks}, need also to be provided.

Atomic Orbitals (AOs) are defined as

$$\chi_i(\mathbf{r}) = \mathcal{M}_i\, P_{\eta(i)}(\mathbf{r})\, R_{\theta(i)}(\mathbf{r})$$

where $\theta(i)$ returns the shell on which the AO is expanded, and $\eta(i)$ denotes which angular function is chosen and P are the generating functions of the given angular momentum $\eta(i)$. Here, the parameter $\mathcal{M}_i$ is an extra parameter which allows the normalization of the different functions of the same shell to be different, as in GAMESS for example.

Molecular orbitals

The calculation of molecular orbitals (MOs) is one such key kernel which can become a bottleneck unless one uses a highly optimized implementation. In this documentation implementation, we have provided an extremely simple algorithm which uses a BLAS Level 3 call to a matrix multiplication routine (`qmckl_dgemm`),

$$\phi_i(r_j) = \sum_k C_{ik} \chi_k(r_j),$$

thus enabling HPC experts to easily identify and optimize the calculation of molecular orbitals. Note that in this pedagogical implementation of the library, the calculation of the Molecular Orbitals (MOs) uses a native call to the `qmckl_dgemm` function.

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.

Slater Determinants

Once the molecular orbitals are implemented, the Slater matrix $S_{ij} = \phi_i(\mathbf{r}_j)$, where $\mathbf{r}_j$ represents the Cartesian coordinates of the j-th electron, needs to be calculated along with its inverse S^{-1}.

Here again we encounter a general function `qmckl_invert` for which we have also provided a small native library. An efficient implementation of the inverse is crucial since the scaling of the algorithm is cubic. We have specially provided hand-coded functions for the calculation of the inverse of small matrices ($M \leq 5$) for which the $\mathcal{O}(N!)$ algorithm is faster than calling external library functions. Efficient methods based on the adjoint of a matrix [4] are used to compute the gradient of the Slater matrix, which is necessary for the computation of the drift vector used in the dynamics of the electrons, and the Laplacian of the Slater matrix required for the computation of the electronic kinetic energy. Both quantities involve the computation of the inverse of the Slater matrix.

For single determinant wave functions, computing the inverse of the Slater matrix translates to a call to `qmckl_invert` function. For multi-determinant wave functions, more sophisticated kernels need to be implemented as presented in the next subsection.

For the computation of the gradient and Laplacian, a special function `qmckl_dgemm_diag` has been implemented and used which provides the diagonal of the result of a matrix product.

Inverse of the Slater matrix

The Sherman-Morrison formula computes the inverse of the sum of a non-singular matrix A of size $n \times n$ and an outer product uv^T of vectors u and v of size n :

$$(A + uv^T) = A^{-1} - \frac{A^{-1}uv^T A^{-1}}{1 + v^T A^{-1} u} \tag{1}$$

This formula is especially useful when one is interested in the inverse of $A+uv^T$ and A^{-1} is already available. Thus, this method only requires to apply a small number of rank-1 updates $A + u_i v_i^T$ to a matrix, and is computationally cheaper than inverting the matrix with standard techniques like LU-decomposition.

This algorithm is the main bottleneck of multi-determinant calculations, so it is crucial to have the most efficient implementation. In addition, this algorithm is known to be responsible for propagating numerical errors, so a careful study of the performance and of the accuracy of multiple kernels was realized, thanks to Verificarlo-CI. The involved algorithms are:

SM1:	The simple Sherman-Morrison formula (Eq. (1))
SM2:	Slagel's splitting method[5]
SM3:	Reordering of the updates
SM4:	Reordering of the updates, then Slagel's splitting
MaponiA3:	Maponi's pivoting technique[6]

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.

4 Applications of QMCkl

The proposed library can already be used in production and for methodological developments. A few independent examples of its first applications are presented in this section.

Jastrow factor in QMC=Chem

The strength of QMC=Chem is large multi-determinant expansions, so very little effort was put in the development of the Jastrow factor. Only the simplest form of Jastrow factor was initially available. As shown above, the sophisticated form of Jastrow factor developed in Champ was introduced in QMCkl, so this functionality has been made quickly available in QMC=Chem by calling the QMCkl appropriate functions.

Sherman-Morrison-Woodbury kernels in QMC=Chem

The Sherman-Morrison-Woodbury formula is the most important kernel in simulations involving large multi-determinant expansions. Hence, it has been carefully optimized in QMC=Chem in the past. The kernels developed in QMCkl can now be called in QMC=Chem in order to compare the performance of the library with QMC=Chem. The comparisons are still ongoing, but is now easy to measure the efficiency of the kernels developed in QMCkl since we can now measure their efficiency beyond benchmark situations, under real simulation conditions.

Exploratory methods using neural networks

In WP4, one project consists in using a neural network as a form of Jastrow factor. Many frameworks for machine learning are in Python or Java, which can't be easily used inside a Fortran program. Moreover, for this particular project we need to evaluate the gradient and Laplacian of the Jastrow factor, which requires the ability to compute derivatives of the neural network. Computing derivatives is particularly easy with the PyTorch package which is for the Python programming language. In this project, we will use Python as the main language, PyTorch for the neural network and QMCkl for the Slater-determinant component. This project will illustrate the versatility of QMCkl, and the importance of providing a library that can be easily used in any programming language.

Exascale-related algorithms

Another project of WP4 is the development of a novel algorithm for Diffusion Monte Carlo (DMC), enabling an efficient load-balancing by making the walkers de-synchronized. In this particular work, it is only necessary to have a "black box" that computes the local energy and the gradient of the wave function, which can be both provided by QMCkl. This work will first be realized in the Julia or in the Python language, which are very good for quick prototyping. Then, once the algorithm is validated on a large scale, it will be implemented in the flagship codes.

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.

Debugging TREXIO files

We expect the TREXIO library to be widely adopted by the community of quantum chemistry. However, writing files in TREXIO format requires the developer to be well aware of the conventions used in the code, namely the normalization factors of the basis functions, the order in which the Cartesian functions appear, etc. It is important to provide a tool with TREXIO, that will allow the developer to check that the file written is valid. A good check is to verify numerically that the molecular orbitals are orthonormal. Passing this test implies that the data relative to the basis set is consistent with the coefficients of the molecular orbitals, so the TREXIO file is valid. We have developed such a tool writing a simple program which calls QMCKL to evaluate the molecular orbitals at grid points and compute a numerical approximation of the overlap matrix in the basis of MOs.

5 Future work

Within TREX, the advances in QMCKL will be twofold. Progress will be made on the development side of the library, by increasing the quality of the documentation of the currently implemented kernels, and by adding more kernels. In parallel, we will start to use more intensively QMCKL within the codes, and the feedback given by the users will drive new improvements.

Development

We have now reached a milestone where we are able to compute the local energy of an arbitrary system described as a single determinant with a Gaussian basis set. Reaching this stage enables the development of HPC variants of the library in WP3, and the exploratory benchmarks to monitor the performance of the kernels on CPUs and GPUs as well as the efficiency in the way they are scheduled. However, there is still a lot of work to do in this WP to make QMCKL usable by external developers.

The most important kernels that still need to be implemented are:

- Kernels related to Effective Core Potentials (ECPs)
- Multi-determinant wave functions
- Different forms of Jastrow factors
- Atomic basis functions input as values on grid points, to enable the use of arbitrary forms of atomic orbitals
- Cusp-fitting of the molecular orbitals at the nuclear positions
- Kernels related to periodic calculations

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.

Applications

We plan in the future to finish the integration of QMCKL into the QMC codes of the CoE. The efficient evaluation of the molecular orbitals is now a performance bottleneck in TURBORVB that will be solved by using the HPC implementations of QMCKL. Similarly, the three-body component of the Jastrow factor in CHAMP will be accelerated by using QMCKL.

Other applications of the library are possible beyond QMC. For instance, the QUANTUM PACKAGE code allows to combine Density Functional Theory (DFT) with wave function methods. DFT methods require the computation of the one-electron density on a grid since many important integrals can only be approximated numerically. We plan to use QMCKL in QUANTUM PACKAGE to accelerate the computation of the density grids to demonstrate the potential of the library beyond QMC simulations.

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] E. Schulte, D. Davison, T. Dye, and C. Dominik, "A Multi-Language Computing Environment for Literate Programming and Reproducible Research," *Journal of Statistical Software*, vol. 46, no. 1, pp. 1–24, Jan 2012. [Online]. Available: https://doi.org/10.18637/jss.v046.i03

[2] "Org Mode," Jan 2021, [Online; accessed 10. Mar. 2021]. [Online]. Available: https://orgmode.org

[3] 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

[4] B. L. Hammond, W. A. Lester, and P. J. Reynolds, *Monte Carlo Methods in Ab Initio Quantum Chemistry*. WORLD SCIENTIFIC, 1994. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/1170

[5] J. T. Slagel, *The Sherman Morrison Iteration. Masters Thesis*. Virginia Polytechnic Institute and State University, 2015.

[6] P. Maponi, "The solution of linear systems by using the sherman–morrison formula," *Linear Algebra and its Applications*, vol. 420, pp. 276–294, 2007.

[7] N. J. Higham, "The accuracy of floating point summation," *SIAM J. Sci. Comput., 14(4), 783–799. (17 pages)*, 1993.

[8] "Kahan summation algorithm - wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Kahan_summation_algorithm

[9] L. Kobbelt, "A fast dot-product algorithm with minimal rounding errors," 1994. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.6693

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.

A Verificarlo-CI tutorial

This tutorial for the CI functionality of Verificarlo is available as a dedicated repository[1]. It describes how to setup the CI pipeline and generate the report on a simple example. Users can set up a CI workflow on the simple code transcribed here.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 4096

float naiveDotprod(float * x, float * y, size_t n) {
        float res = 0;
        for(size_t i=0; i<n; i++) {
                res += x[i] * y[i];
        }
        return res;
}

int main(void) {
        float x, y [N];
        srand(42);
        for(size_t i=0; i<N; i++) {
                x[i] = (float) rand() / RAND_MAX;
                y[i] = (float) rand() / RAND_MAX;
        }
        srand(time(NULL));

        float naiveRes = naiveDotprod(x, y, N);
        printf("Naive dotprod = %.7f \n", naiveRes);

        return 0;
}
```

The goal of this example is to compute a basic dot product on two vectors generated with a fixed with random seed. The computation is done with a naive method, and the function that does it can be found at the beginning of `main.c`. The goal will be to use Verificarlo CI to measure the numerical accuracy of the naive implementation, before adding a better version of the algorithm and validating it by comparing the results of the two versions.

Following this tutorial will require you to have a functional install of Verificarlo on your machine, or to use it through its Docker image. For more information on the tool itself, please refer to the Verificarlo CI documentation. It is advised that you at least go through it quickly before doing this tutorial.

[1] https://github.com/verificarlo/vfc_ci_tutorial

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.

Fork and clone the repository

In order to follow this tutorial, which makes use of GitHub Actions, you will need to have your own fork of this repository. Once it is created, `git clone` it:

```
$ git clone https://github.com/[YourUserName]/verificarlo_ci_tutorial
$ cd verificarlo_ci_tutorial
```

Try to build and execute the code

Here is the output that you should get after issuing the following commands:

```
$ make
$ ./dotprod
Naive dotprod = 1040.9039307
```

Add your first test probes

`vfc_probes` is the system used by Verificarlo CI to export test variables from a program to the tool itself. First of all, you'll need to modify the Makefile to link the `vfc_probes` library. Line 4 should become:

```
all:
        $(CC) main.c -lvfc_probes -o dotprod
```

Moreover, you must also include the `vfc_probes.h` header at the beginning of `main.c`. You should add the following include statement after line 3:

```
#include <vfc_probes.h>
```

Now that `vfc_probes` is correctly linked, we can create out probes structure. This should be done at the beginning at the main function, for instance after the new line 20 (if you added the previous line):

```
int main(void) {
  vfc_probes probes = vfc_init_probes();
```

Then we will add the probe containing the result of the `dotprod`. You can do this after the new line 32:

```
  float naiveRes = naiveDotprod(x, y, n);
  vfc_probe(&probes, "dotprod_test", "naive", naiveRes);
```

Finally, we can dump the probes at the end of the main function, just before the return statement:

```
  vfc_dump_probes(&probes);
  return 0;
```

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.

Set up `vfc_ci` and GitHub Actions

Before setting up GitHub Actions, we need to make sure that the `vfc_ci` test command can run. To do this, create `vfc_tests_config.json`, the test configuration file, at the root of the repository and with the following content:

```
{
    "make_command": "make CC=verificarlo-c",
    "executables": [
        {
            "executable": "dotprod",
            "vfc_backends": [
                {
                    "name": "libinterflop_mca.so",
                    "repetitions": 20
                },
                {
                    "name": "libinterflop_mca.so --mode=rr",
                    "repetitions": 20
                }
            ]
        }
    ]
}
```

Each test run will consist of 40 executions of the dotprod executable, split over two backends. This will export one test probe containing the result of the naive dotprod. To make sure that everything works correctly, it is possible to call `vfc_ci` test with the dry run flag, so that no output file will be produced:

```
$ vfc_ci test -d
[...]
Info [vfc_ci]: The results have been successfully written to XXXXXXXXXX.vfcrun.h5.
Info [vfc_ci]: The dry run flag was enabled, so no files were actually created.
```

If you get the following output, your Verificarlo CI setup should be correct.

Before setting up the CI workflow, we need to make sure that our local repository doesn't contain any unstaged changes. Commit and push the changes that you have just made:

```
$ git add .
$ git commit
$ git push --set-upstream origin master
```

You are finally ready to create the CI workflow. This can be done automatically with the following command:

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.

```
$ vfc_ci setup github
Info [vfc_ci]: A Verificarlo CI workflow has been setup on master.
Info [vfc_ci]: Make sure that you have a "vfc_tests_config.json" on this branch.
You can also perform a "vfc_ci test" dry run before pushing other commits.
```

This should create a commit on the master branch (which we will call the dev branch), as well as a `vfc_ci_master` branch (which we will call the CI branch). A first test run will also be triggered immediately after the commit, which will result in a test file being committed to the CI branch.

Serve the test report

Once the test file has been committed to the CI branch, you can checkout to it and access the results:

```
$ git checkout vfc_ci master
$ git pull origin vfc_ci_master
$ cd vfcruns
$ vfc_ci test -s
```

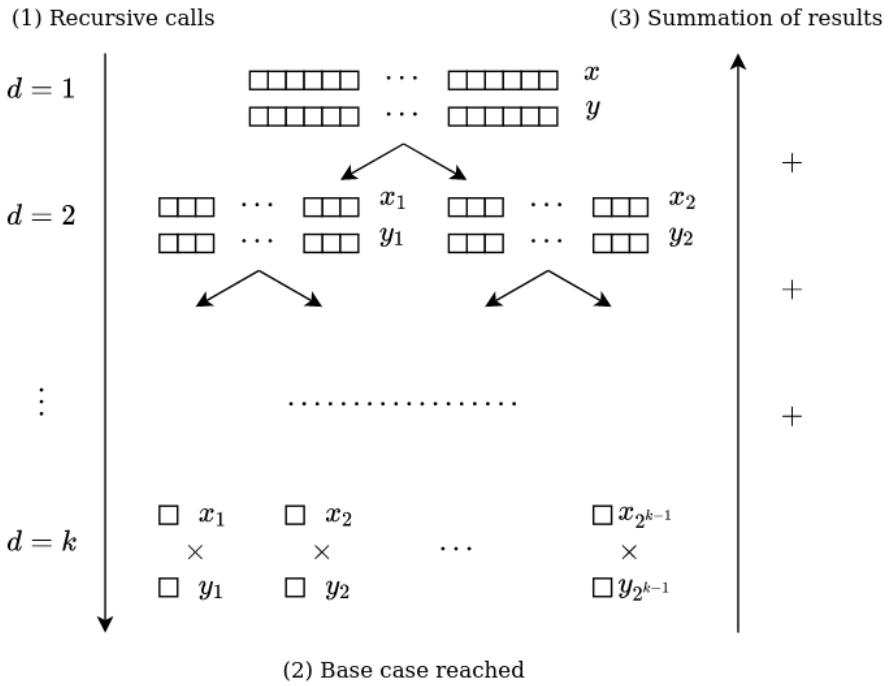
. . . will open the report in your browser.

There's not much to see for now, since there's only one test variable and one run in the report. However, if the naive algorithm could seem sufficient at first glance, it does have one issue. The values of `x[i] * y[i]` are all added to the same `res` variable, which will suffer from increasingly important round off errors. Thus, when a cancellation occurs, it will become catastrophic and yield an inaccurate result. It has been shown that this error grows in $O(n)$.

Adding an improved method

This subsection introduces another method for the computation of a dot product. The idea is to recursively split the `x` and `y` arrays in half as to obtain a binary tree, and to compute `x[i] * y[i]` when the arrays are of size 1 (which is the base case). The final result is computed by adding for each node the dot products of its two children, until the root is reached. This method is called pairwise summation and is illustrated in the figure 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.



Since the results of each layer are added together instead of being accumulated into one term, error will increase after each layer of the tree, instead of after each operation, thus growing in $O(\log n)$ as shown in [7].

Here is the implementation of the algorithm that you can add before the main function:

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.

```
float recursiveDotprod(float * x, float * y, size_t n) {
        // NOTE This implementation assumes that N can be written as 2^k
        if((n & (n - 1)) != 0) {
                return 0;
        }
        // Base case
        if(n == 1) {
                return x[0] * y[0];
        }
        // Recursive case
        else {
                // Split array in 2 and do a recursive call for each half
                return recursiveDotprod(x, y, n / 2) +
                        recursiveDotprod(&(x[n/2]), &(y[n/2]), n / 2);
        }
}
```

Call the function in `main.c` and add a new probe:

```
float recursiveRes = recursiveDotprod(x, y, n);
vfc_probe(&probes, "dotprod_test", "recursive", recursiveRes);
```

...and optionally a `printf` statement:

```
printf("Naive dotprod = %.7f \n", naiveRes);
printf("Recursive dotprod = %.7f \n", recursiveRes);
```

Finally, commit and push the changes to your remote repository. Optionally, you can also run `vfc_ci test` yourself to generate a results file and directly generate the report with `vfc_ci serve` (in which case only the results you have just generated will appear, since the first results file is not in your work tree).

Using the report to compare the two algorithms

Checkout to the CI branch and launch the test report. A second commit including the new test probe should appear. In the "Inspect run" section, which allows to examine and compare results from a single run, we should be able to compare the two algorithms:

The following plots seem to validate our assumptions about the recursive algorithm: the recursive version has both a higher number of significant digits (around 6.9 vs 5.9 with `libinterflop_mca.so`) and a lower standard deviation with both backends.

In this simple example, `vfc_ci` allowed us to quickly compare two algorithms and confirm that our second one is numerically more stable than the naive version. Of course, the proposed solution does not aim to be optimal, and you can try to implement other methods such as the Kahan summation [8] or Kobbelt's dot product algorithm [9].

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.

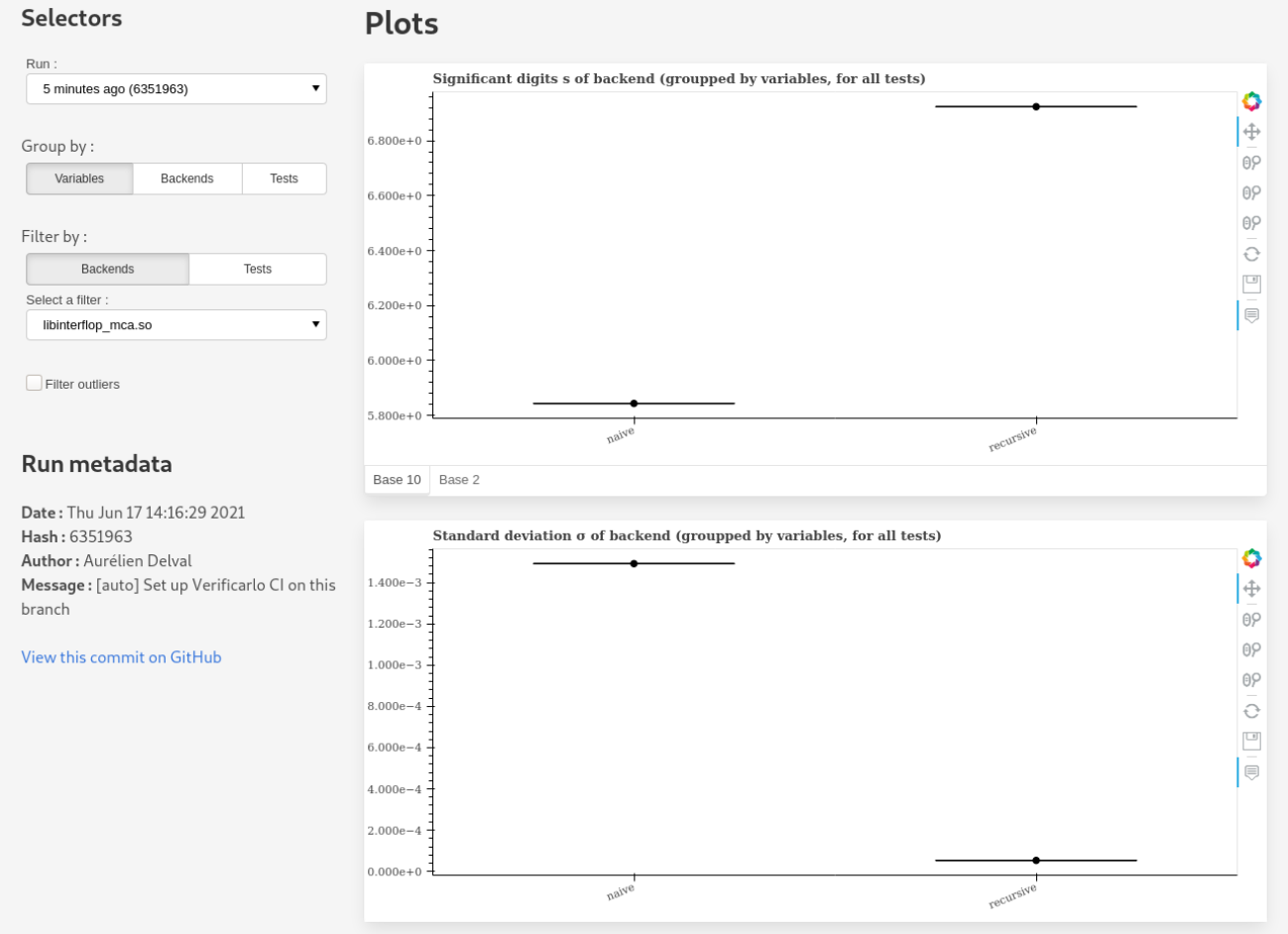


Figure 2: Comparison of both `dotprod` algorithms

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.