Targeting Real chemical accuracy at the EXascale

TREX : an innovative view of HPC usage applied to Quantum Monte Carlo simulations

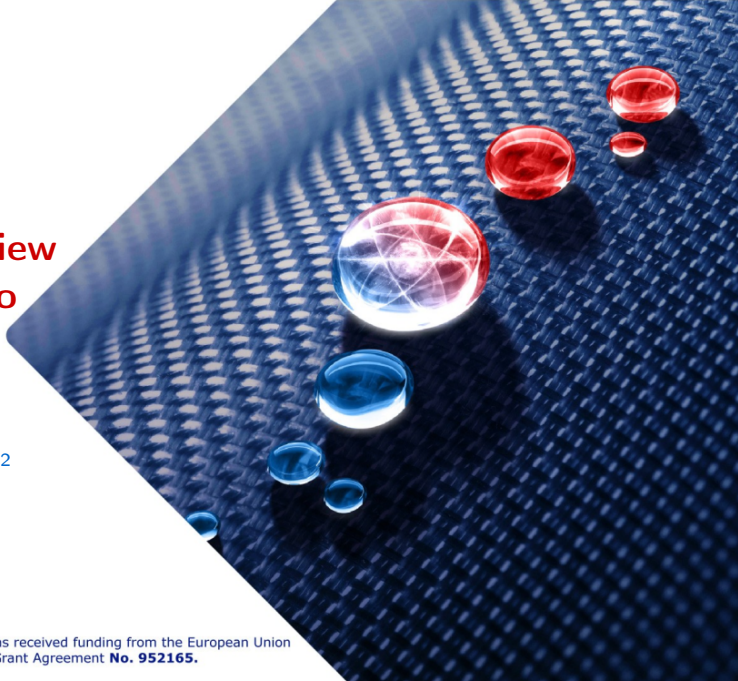
Anthony Scemama[1], Pablo de Oliveira Castro[2], Cedric Valensi[2], William Jalby[2]

02/07/2021

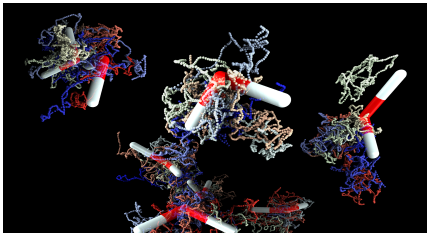
[1]University of Toulouse/CNRS, LCPQ (France)

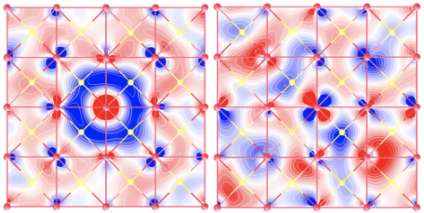
[2]University of Versailles, Li-PaRAD (France)

Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizoon 2020 research and innovation programme under Grant Agreement **No. 952165.**



- Describing matter with quantum mechanics (Schrödinger's equation)
- Users: theoretical chemists and physicists





Implications for society

- Health	Drug design
- Electronics	Nano- and micro-electronics
- Materials	Carbon nanotubes, graphene, ...
- Catalysis	Enzymatic reactions, petroleum

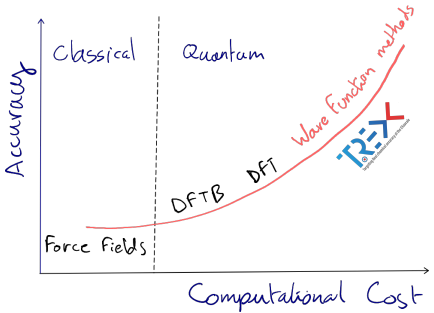
Partners

UNIVERSITY OF TWENTE.
cnrs
CINECA
JÜLICH Forschungszentrum
MAX-PLANCK-GESELLSCHAFT
MEGWARE
Politechnika Łódzka
SISSA
STU SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
Trust-IT Services Communicating ICT to markets
universität wien
UVSQ université PARIS-SACLAY

Codes

- CHAMP
- QMC=Chem
- TurboRVB
- NECI
- Quantum Package
- GammCor

TREX: Targeting REal chemical accuracy at the EXascale



Objective: Make codes ready for exascale

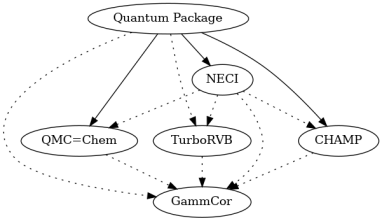
How: Instead of re-writing codes, provide libraries

- A library for exchanging information between codes (TREXIO) $\Longrightarrow$ Enables HTC
- A library for high-performance (QMCkl) $\Longrightarrow$ Enables HPC

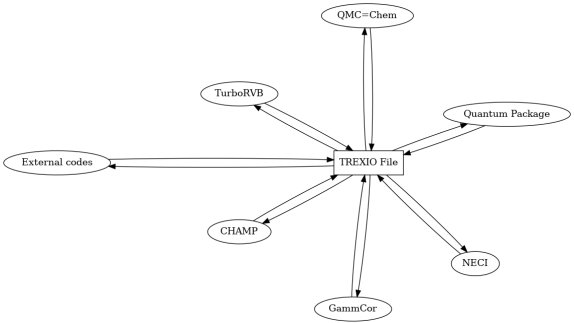
QMC: Quantum Monte Carlo methods

- Highly accurate
- Massively parallelisable (multiple QMC trajectories)
- CPU intensive

Before



After



(BSD license)

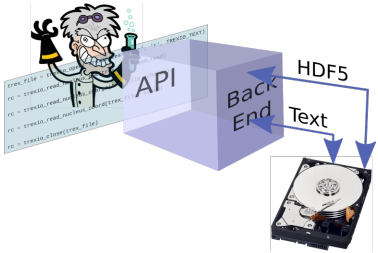
`https://github.com/trex-coe/trexio`

Front end

- Definition of an API for to read/write wave functions
- C-compatible API: Easy bindings in other languages

Content of the files

- File is self-contained: no external knowledge needed to compute $\Psi(r_1, \ldots, r_n)$ (normalization factors, basis et parameters, *etc*)
- Strong conventions (atomic units, ordering of cartesian orbitals, *etc*)



Back end

- HDF5: Efficient I/O
- Text: debugging, fallback when HDF5 can't be installed

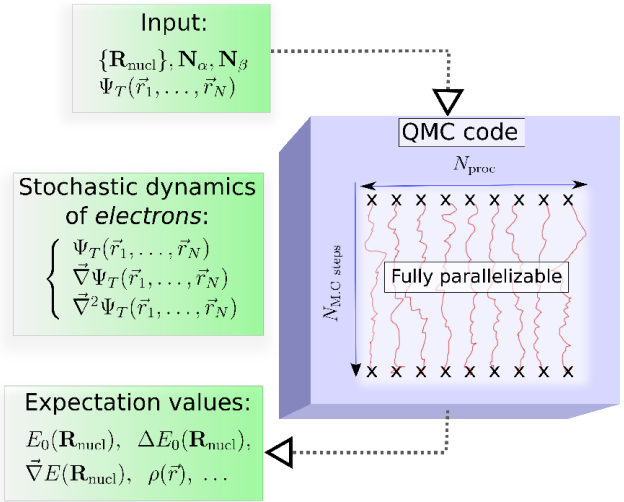
Source code generated from a config file.

Problem: Stochastic resolution of the Schrödinger equation for N electrons

$$
\begin{aligned}
E &= \frac{\int \mathrm{d}\mathsf{r}_1 \dots \mathrm{d}\mathsf{r}_N \, \Phi(\mathsf{r}_1, \dots, \mathsf{r}_N) \mathcal{H} \Phi(\mathsf{r}_1, \dots, \mathsf{r}_N)}{\int \mathrm{d}\mathsf{r}_1 \dots \mathrm{d}\mathsf{r}_N \, \Phi(\mathsf{r}_1, \dots, \mathsf{r}_N) \Phi(\mathsf{r}_1, \dots, \mathsf{r}_N)} \\
&\sim \sum \frac{\mathcal{H}\Psi(\mathsf{r}_1, \dots, \mathsf{r}_N)}{\Psi(\mathsf{r}_1, \dots, \mathsf{r}_N)}, \text{ sampled with } (\Psi \times \Phi)
\end{aligned}
$$

- $\mathcal{H}$: Hamiltonian operator
- E: Energy
- $\mathsf{r}_1, \dots, \mathsf{r}_N$: Electron coordinates
- Φ: Almost exact wave function
- Ψ: Trial wave function

- Very low memory requirements (no integrals)
- Distribute walkers on different cores or compute nodes
- No blocking communication: near-ideal scaling
- Difficulty: parallelize within a QMC trajectory



Computational kernels

- QMCkl will contain the main kernels of QMC methods
- Written together by QMC experts and HPC experts
- Multiple high performance implementations of the kernels, tuned for different
  - architectures
  - problem sizes
  - requested accuracy (reduced precision)

Two implementations

- *Documentation* : easy to read and understand, not necessarily efficient
- *High performance* : efficient, but not necessarily readable by physicists/chemists
- Both *Documentation* and *High performance* have the same API.

Advantages

- The code can stay easy to understand by the physicists/chemists
  Performance-related aspects are delegated to the library
- Scientists can use their preferred language
- Scientists don't lose control on their codes
- Codes don't die when the architecture changes
- Scientific code development does not break the performance
- Better re-use of the optimization effort among the community

- Same API as the documentation library
- Optimization is guided by analysis with MAQAO[1].
- Propose performance-critical choices in the API design (data structures, memory management, *etc*)
- Both CPU and GPU versions of the kernels
- Task parallelism with StarPU[2] to schedule kernels on CPU and GPU and handle asynchronous CPU-GPU transfers

[1] https://maqao.org
[2] C. Augonnet et al, doi:10.1002/cpe.1631

MAQAO support to the developer

- Identify profitable optimizations (partial/full vectorization, data access restructuring, blocking/interchanging, load balancing etc.…)
- Perform a Return on Investment (ROI) analysis to help the developer select the most profitable optimization

Unicore run on TURBO RVB
(S. Sorella:SISSA)

Global Metrics

Total Time (s)		481.84
Profiled Time (s)		481.84
Time in analyzed loops (%)		18.51
Time in analyzed innermost loops (%)		13.1
Time in user code (%)		25.35
Compilation Options		OK
Perfect Flow Complexity		1.01
Array Access Efficiency (%)		83.97
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.06
	Nb Loops to get 80%	12
FP Vectorised	Potential Speedup	1.04
	Nb Loops to get 80%	12
Fully Vectorised	Potential Speedup	1.17
	Nb Loops to get 80%	19
FP Arithmetic Only	Potential Speedup	1.11
	Nb Loops to get 80%	16

Extensive/automatic testing of different configurations

Comparative analysis

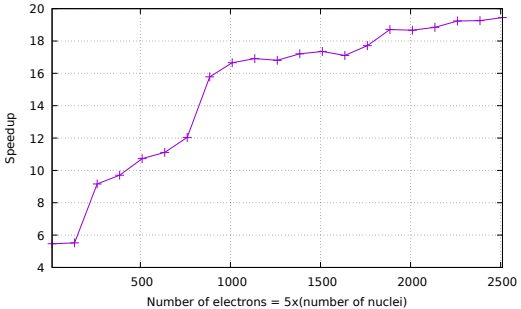
- Automatically perform comparative runs to analyze impact of compiler, dataset, algorithm and parallel configuration (number of cores, etc..)
- Analysis can be performed daily or weekly

r1: 1 core r2: 2cores r3: 4 cores r4: 8 cores r5: 16 cores r6: 32 cores r7: 52 cores. Multicore runs on TURBO RVB (S. Sorella. SISSA)

Global Metrics

Metric		r1	r2	r3	r4	r5	r6	r7
Total Time (s)		555.66	292.81	156.88	88.89	63.01	56.46	52.85
Profiled Time (s)		555.66	292.81	156.88	88.89	63.01	56.46	52.85
Time in analyzed loops (%)		43.0	41.7	38.7	34.3	29.3	22.6	16.6
Time in analyzed innermost loops (%)		37.9	36.7	34.0	29.8	26.1	20.6	15.3
Time in user code (%)		49.7	47.9	45.0	40.0	33.4	25.3	18.6
Compilation Options		OK	OK	OK	OK	OK	OK	OK
Perfect Flow Complexity		1.00	1.00	1.00	1.00	1.00	1.00	1.00
Array Access Efficiency (%)		92.4	92.1	92.0	91.7	92.7	93.3	91.6
Perfect OpenMP + MPI + Pthread		1.00	1.02	1.04	1.05	1.11	1.14	1.31
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00	1.03	1.09	1.18	1.35	1.76	2.37
No Scalar Integer	Potential Speedup	1.06	1.05	1.05	1.05	1.03	1.02	1.02
	Nb Loops to get 80%	13	13	13	13	13	13	13
FP Vectorised	Potential Speedup	1.02	1.02	1.02	1.02	1.01	1.01	1.01
	Nb Loops to get 80%	2	3	3	3	2	2	2
Fully Vectorised	Potential Speedup	1.35	1.34	1.31	1.27	1.21	1.15	1.11
	Nb Loops to get 80%	12	13	13	12	11	9	8
Only FP Arithmetic	Potential Speedup	1.10	1.10	1.10	1.09	1.06	1.04	1.03
	Nb Loops to get 80%	16	17	17	16	16	17	17
OpenMP perfectly balanced	Potential Speedup	1.00	1.01	1.05	1.06	1.07	1.14	1.10
	Nb Loops to get 80%	1	5	3	4	5	4	6

$$J_{\text{een}}(\mathrm{r},\mathrm{R}) = \sum_{\alpha=1}^{N_{\text{nucl}}} \sum_{i=1}^{N_{\text{elec}}} \sum_{j=1}^{i-1} \sum_{p=2}^{N_{\text{nord}}} \sum_{k=0}^{p-1} \sum_{l=0}^{p-k-2\delta_{k,0}} c_{lkp\alpha} \, (r_{ij})^k \left[(R_{i\alpha})^l + (R_{j\alpha})^l \right] (R_{i\,\alpha} \, R_{j\alpha})^{(p-k-l)/2}$$



- Gradient and Laplacian are also required
- Up to 20× faster than in the original code
- $\sim$ 80% of the AVX-512 peak is reached
- Expressed with a DGEMM kernel $\Longrightarrow$ also efficient on GPU

Verificarlo is a tool for assessing the precision of floating point operations. It can be used to :

`https://github.com/verificarlo/verificarlo`
GPL v3

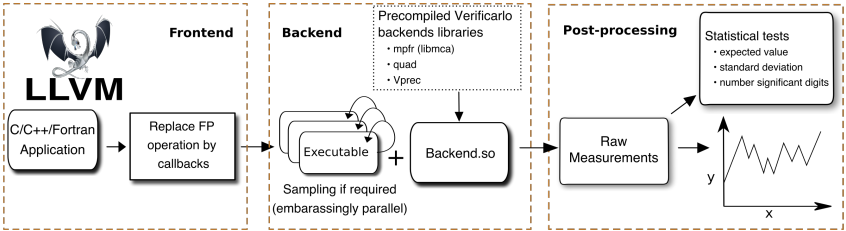
- **Find numerical bugs** in codes [1]
  - Stochastic arithmetic to simulate round-off and cancellations
  - Localization techniques to pinpoint source of errors
- **Optimize precision** [2]
  - Simulate custom formats for mixed precision (float, bf16)
  - Tune precision in math library calls

[1] C. Denis *et al.* doi:10.1109/ARITH.2016.31
[2] Y Chatelain *et al.* doi:10.1007/978-3-030-29400-7_34

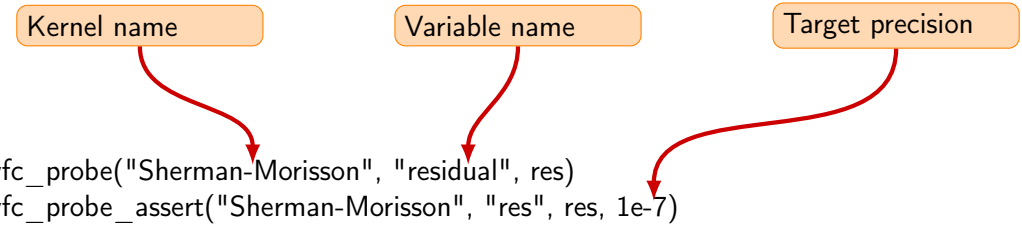
- Each Floating-Point (FP) operation may introduce a δ error

$$z = fl[x + y] = (x + y)(1 + \delta)$$

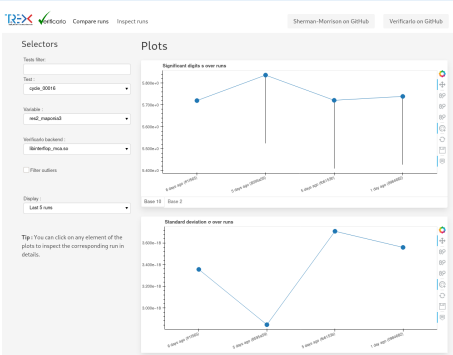
- When chaining multiple operations, errors can accumulate and snowball
- Monte Carlo Arithmetic key principle
  - Make δ a random variable
  - Use a Monte Carlo simulation to empirically estimate the FP error distribution



- Each push to QMCkl triggers a Verificarlo analysis.
- QMCkl kernels unit tests are augmented with probes:
  - track a scalar value precision
  - ensure that a target precision is reached

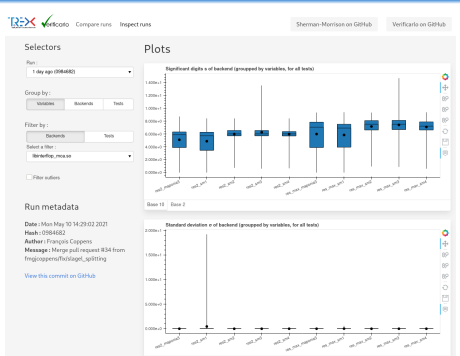


Compare runs



- Track precision of kernels over commits
- Shows significant digits s, standard deviation σ, variable distribution

Inspect runs



- Focus in depth on one particular run
- Compare multiple implementations of the same kernel

Useful links

TREX web site	`https://trex-coe.eu`
TREXIO	`https://github.com/trex-coe/trexio`
QMCkl	`https://github.com/trex-coe/qmckl`
QMCkl documentation	`https://trex-coe.github.io/qmckl`
MAQAO	`http://www.maqao.org`
Verificarlo	`https://github.com/verificarlo/verificarlo`