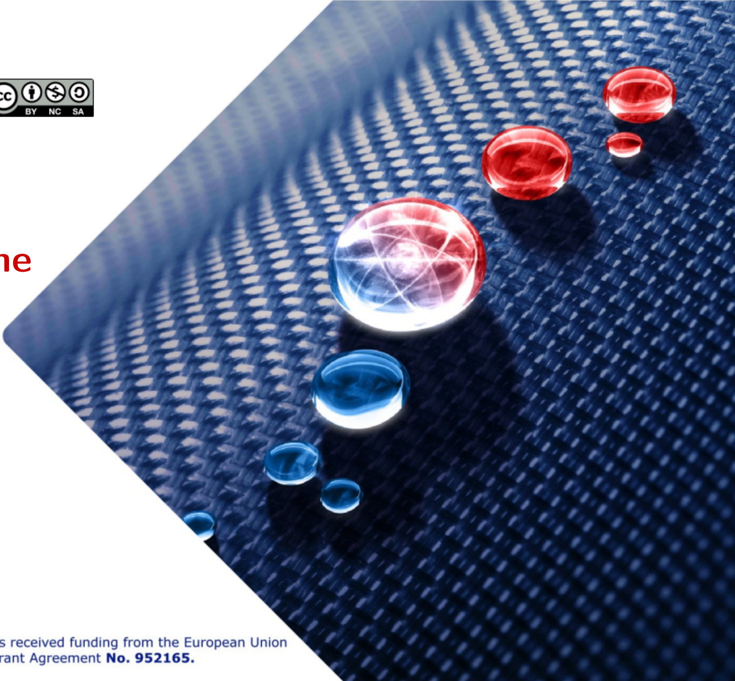
Libraries developed in the TREX CoE

A. Scemama[1], V.G. Chilkuri[1], E. Posenitskiy[1], P. de Oliveira Castro[2], C. Valensi[2], W. Jalby[2]

04/10/2022

[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.**

Quantum Monte Carlo in TREX

QMC: Quantum Monte Carlo methods

- Highly accurate electronic structure methods (solids and molecules)
- Massively parallelisable (multiple QMC trajectories)
- Very CPU intensive: One of the most "compute-hungry" methods
- Still under development: scientists need to run *and* develop code
- Input data is complex (electronic wave function)

Objective: Make codes ready for exascale

How: Instead of re-writing codes, provide libraries (free software)

1. TREXIO: A library for exchanging information between codes $\Longrightarrow$ Enables HTC
2. QMCkl: A library for high-performance $\Longrightarrow$ Enables HPC

Quantum Monte Carlo (QMC)

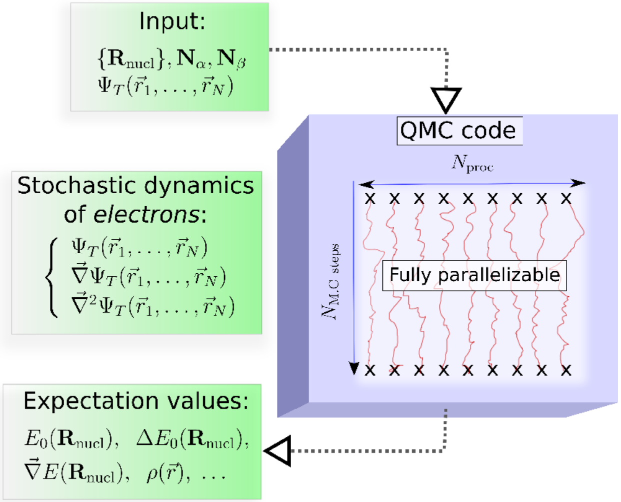
Problem: Stochastic resolution of the Schrödinger equation for N electrons

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

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

Quantum Monte Carlo (QMC)

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



Input:
$\{\mathbf{R}_{\text{nucl}}\}, \mathbf{N}_\alpha, \mathbf{N}_\beta$
$\Psi_T(\vec{r}_1, \dots, \vec{r}_N)$
QMC code
N_{proc}
$N_{\text{M.C steps}}$
Fully parallelizable
Stochastic dynamics of *electrons*:
$\Psi_T(\vec{r}_1, \dots, \vec{r}_N)$
$\vec{\nabla}\Psi_T(\vec{r}_1, \dots, \vec{r}_N)$
$\vec{\nabla}^2\Psi_T(\vec{r}_1, \dots, \vec{r}_N)$
Expectation values:
$E_0(\mathbf{R}_{\text{nucl}}), \Delta E_0(\mathbf{R}_{\text{nucl}}),$
$\vec{\nabla}E(\mathbf{R}_{\text{nucl}}), \rho(\vec{r}), \dots$

Both libraries

Three objectives

1. Productivity
   Usable and useful by scientists in different programming languages
2. Portability
   Target: all HPC systems (CPU, GPU, ARM, x86, etc.)
3. Performance
   Must be efficient on all architectures: possible tradeoffs between portability and performance

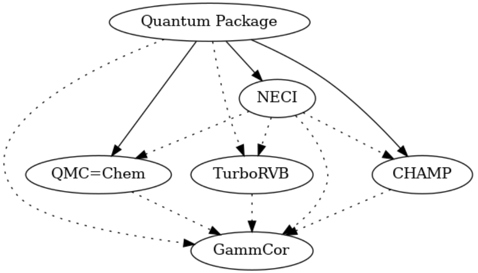
Free (libre) software

- Requirement for open science
- BSD license for adoption by any software (academic, commercial, . . .)

TREXIO: I/O library

TREXIO: I/O library

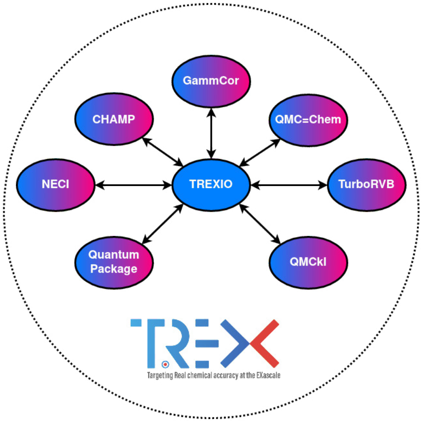
Before



(BSD license)

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

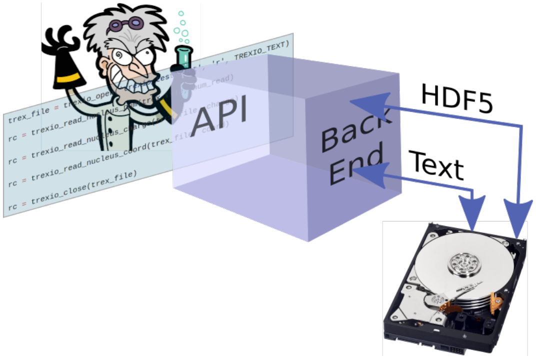
After



TREXIO: I/O library

Front end

- Definition of an API for to read/write wave functions
- C-compatible API: Easy usage in all common languages



```
trex_file = trexio_open('trexio_test[illegible]', 'r', TREXIO_TEXT)
rc = trexio_read_nucleus_num(trex_file, num_read)
rc = trexio_read_nucleus_charge(trex_file, charge)
rc = trexio_read_nucleus_coord(trex_file, coord)
rc = trexio_close(trex_file)
```

Back end

- HDF5: Efficient I/O
- Text:
  - Fallback when HDF5 can't be installed
  - Debugging
  - Version control systems

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 atomic orbitals, etc.)
- The data stored in the files is organized in different groups:

Metadata	Electron	Slater Determinants
Nucleus	Basis	CI coefficients
AO	MO	Two-electron integrals
One-electron integrals	Density matrices	ECP

- Each group contains multiple attributes: information related to the group

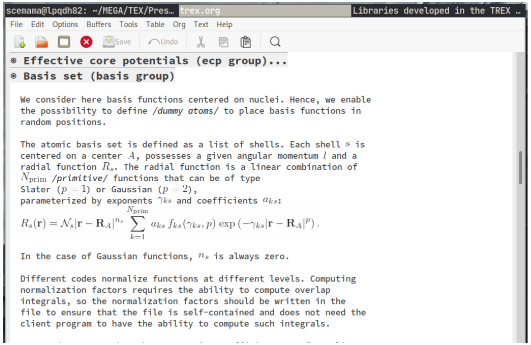
Source code

- For each attribute :

```
trexio_exit_code  trexio_[has|read|write]_<group>_<attribute>
                                          (trexio_t* file, <type> attribute)
```

- The library can be auto-generated by a script as the function names can be computed
- Productivity : Literate programming with Org-mode
  Table $\rightarrow$ JSON $\rightarrow$ C source code
  $\rightarrow$ Documentation
- Fortran, Python/Numpy, and OCaml interfaces are also generated
- Performance : HDF5 back end
- Portability : Only optional dependency is HDF5

Productivity:


scemama@lpqdh82: ~/MEGA/TEX/Pres… trex.org Libraries developed in the TREX …
File Edit Options Buffers Tools Table Org Text Help
Save Undo

⊛ **Effective core potentials (ecp group)...**
⊛ **Basis set (basis group)**

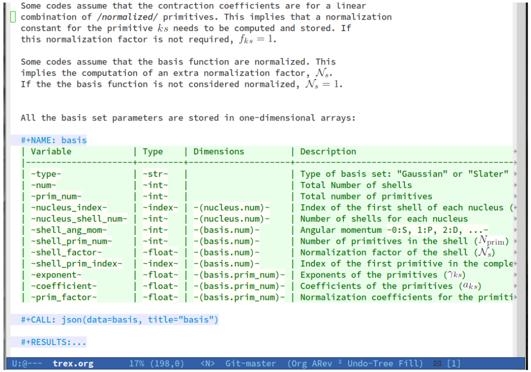
We consider here basis functions centered on nuclei. Hence, we enable the possibility to define */dummy atoms/* to place basis functions in random positions.

The atomic basis set is defined as a list of shells. Each shell s is centered on a center A, possesses a given angular momentum l and a radial function R_s. The radial function is a linear combination of N_{prim} */primitive/* functions that can be of type Slater $(p=1)$ or Gaussian $(p=2)$, parameterized by exponents γ_{ks} and coefficients a_{ks}:

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

In the case of Gaussian functions, n_s is always zero.

Different codes normalize functions at different levels. Computing normalization factors requires the ability to compute overlap integrals, so the normalization factors should be written in the file to ensure that the file is self-contained and does not need the client program to have the ability to compute such integrals.



Some codes assume that the contraction coefficients are for a linear combination of */normalized/* primitives. This implies that a normalization constant for the primitive ks needs to be computed and stored. If this normalization factor is not required, $f_{ks}=1$.

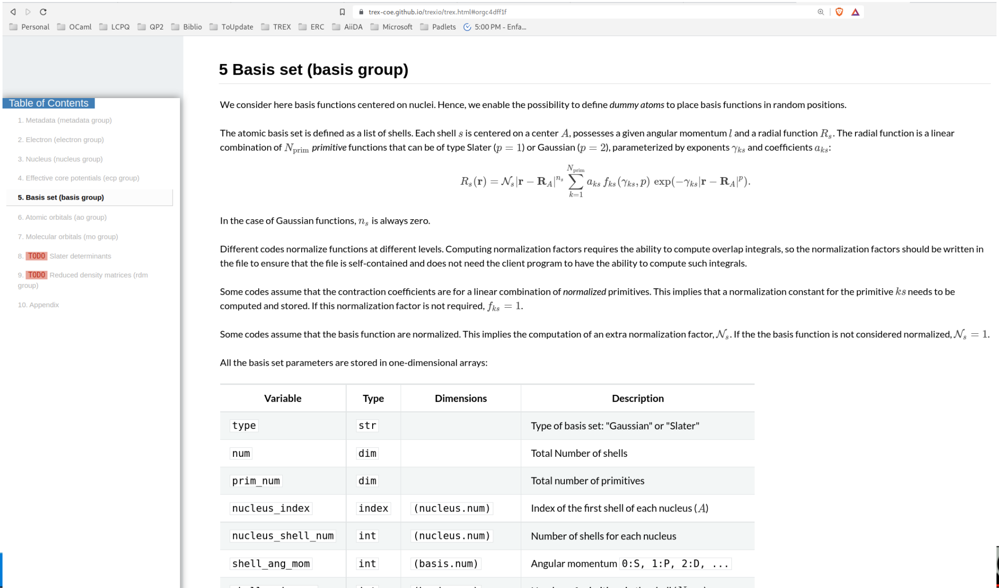
Some codes assume that the basis function are normalized. This implies the computation of an extra normalization factor, $\mathcal{N}_s$. If the the basis function is not considered normalized, $\mathcal{N}_s=1$.

All the basis set parameters are stored in one-dimensional arrays:

```
#+NAME: basis
| Variable            | Type    | Dimensions          | Description
|---------------------+---------+---------------------+------------------------------------------
| ~type~              | ~str~   |                     | Type of basis set: "Gaussian" or "Slater"
| ~num~               | ~int~   |                     | Total Number of shells
| ~prim_num~          | ~int~   |                     | Total number of primitives
| ~nucleus_index~     | ~index~ | ~(nucleus.num)~     | Index of the first shell of each nucleus (
| ~nucleus_shell_num~ | ~int~   | ~(nucleus.num)~     | Number of shells for each nucleus
| ~shell_ang_mom~     | ~int~   | ~(basis.num)~       | Angular momentum ~0:S, 1:P, 2:D, ...~
| ~shell_prim_num~    | ~int~   | ~(basis.num)~       | Number of primitives in the shell ($N_{\text{prim}}$)
| ~shell_factor~      | ~float~ | ~(basis.num)~       | Normalization factor of the shell ($\mathcal{N}_s$)
| ~shell_prim_index~  | ~index~ | ~(basis.num)~       | Index of the first primitive in the comple
| ~exponent~          | ~float~ | ~(basis.prim_num)~  | Exponents of the primitives ($\gamma_{ks}$)
| ~coefficient~       | ~float~ | ~(basis.prim_num)~  | Coefficients of the primitives ($a_{ks}$)
| ~prim_factor~       | ~float~ | ~(basis.prim_num)~  | Normalization coefficients for the primiti

#+CALL: json(data=basis, title="basis")

#+RESULTS:...
```

```
U:@---  trex.org      17% (198,0)   <N>  Git-master  (Org ARev ² Undo-Tree Fill)  [1]
```

Documentation



trex-coe.github.io/trexio/trex.html#orgc4dff1f

Personal OCaml LCPQ QP2 Biblio ToUpdate TREX ERC AiiDA Microsoft Padlets 5:00 PM - Enfa...

Table of Contents

5 Basis set (basis group)

We consider here basis functions centered on nuclei. Hence, we enable the possibility to define *dummy atoms* to place basis functions in random positions.

The atomic basis set is defined as a list of shells. Each shell s is centered on a center A, possesses a given angular momentum l and a radial function R_s. The radial function is a linear combination of N_{prim} *primitive* functions that can be of type Slater ($p = 1$) or Gaussian ($p = 2$), parameterized by exponents γ_{ks} and coefficients a_{ks}:

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

In the case of Gaussian functions, n_s is always zero.

Different codes normalize functions at different levels. Computing normalization factors requires the ability to compute overlap integrals, so the normalization factors should be written in the file to ensure that the file is self-contained and does not need the client program to have the ability to compute such integrals.

Some codes assume that the contraction coefficients are for a linear combination of *normalized* primitives. This implies that a normalization constant for the primitive ks needs to be computed and stored. If this normalization factor is not required, $f_{ks} = 1$.

Some codes assume that the basis function are normalized. This implies the computation of an extra normalization factor, $\mathcal{N}_s$. If the the basis function is not considered normalized, $\mathcal{N}_s = 1$.

All the basis set parameters are stored in one-dimensional arrays:

Variable	Type	Dimensions	Description
`type`	`str`		Type of basis set: "Gaussian" or "Slater"
`num`	`dim`		Total Number of shells
`prim_num`	`dim`		Total number of primitives
`nucleus_index`	`index`	`(nucleus.num)`	Index of the first shell of each nucleus (A)
`nucleus_shell_num`	`int`	`(nucleus.num)`	Number of shells for each nucleus
`shell_ang_mom`	`int`	`(basis.num)`	Angular momentum `0:S, 1:P, 2:D, ...`

QMCkl: QMC kernel library

QMC kernel library

Computational kernels

- QMCkl contains the main kernels of QMC methods: Domain specific library, end-user driven
- Written together by QMC experts and HPC experts
- Multiple high performance implementations of the kernels, tuned for different
  - architectures: portability is critical for users
  - problem sizes: from small to large systems
  - requested accuracy: reduced precision

Objectives

- The code must stay easy to understand by the physicists/chemists. Performance-related aspects should be delegated to the library
- Scientists should be able to use their preferred language
- Scientists should not lose control of their codes
- Codes should not die when the architecture changes
- Scientific code development should not kill the performance
- Reuse of the optimization effort among the community

Functionality and performance

- Keeping high *productivity*, *portability* and *performance* is very hard in a single piece of software.

  We propose (at least) two implementations:

  1. Documentation library
     Easy to read, understand, modify for scientists, not necessarily efficient.
  2. High performance libraries
     Efficient on a given architecture, but not necessarily readable by physicists/chemists.
     Performance within 10% to maximize portability and simplicity.
  3. Ultra-High performance libraries
     Generated with auto-tuning tools for well identified datasets.

- Both *Documentation* and *High performance* have the same API (similar to BLAS on netlib *vs* MKL).
- Scientific progress is made in the documentation library, and implemented in the HPC versions when the API is stabilized.
- Performance: enable a data-driven task-based parallelism

Library design

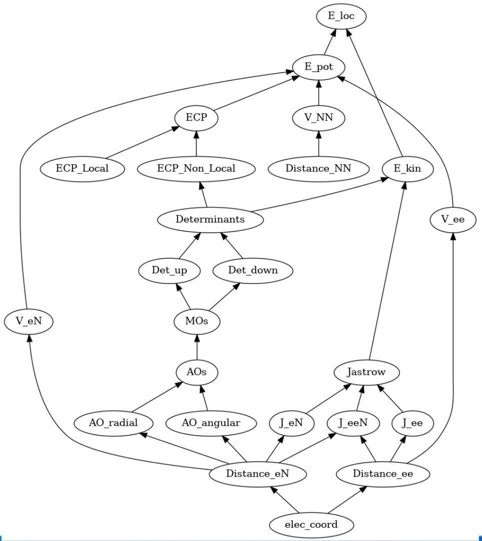
- Creation of a *Context* that keeps a consistent state of the library (pointers to computed data, configuration parameters, etc.)
- Memory allocation is abstract:

```c
void* qmckl_malloc(qmckl_context context, const qmckl_memory_info_struct info);
```

  allows allocation on CPU/GPU by the HPC variants
- Low level functions: access to simple low-level functions leaving the context untouched (no allocation, no modification in-place)
- High-level functions: let the library call multiple kernels in an optimal way, possibly updating the context
- Use of IRP programming paradigm[1] to keep track of dependencies between kernels: re-compute only what is necessary and store computed data in the context

[1] http://arxiv.org/abs/0909.5012

Dependencies between kernels



- Only the needed sub-graph is computed
- HPC: Each kernel is one/many parallel Task(s)
- HPC: Use OpenMP tasks or StarPU for hybrid architectures: (StarPU handles very well asynchronous CPU-GPU transfers).

Use case: low-level

```c
#include <qmckl.h>

// ...
int64_t  m, n, LDA, LDB, LDC;
// ...
double   A[LDA*3];
double   B[LDB*3];
double   C[LDC*n];
// ...

qmckl_context context = qmckl_context_create();

// Compute inter-particle distances between xyz coordinates in A[m][3] and B[3][n]
// and store the result in C[m][n]
qmckl_exit_code rc = qmckl_distance(context, 'N', 'T', m, n, A, LDA, B, LDB, C, LDC);
assert (rc == QMCKL_SUCCESS);
// ...
```

Use case: high-level

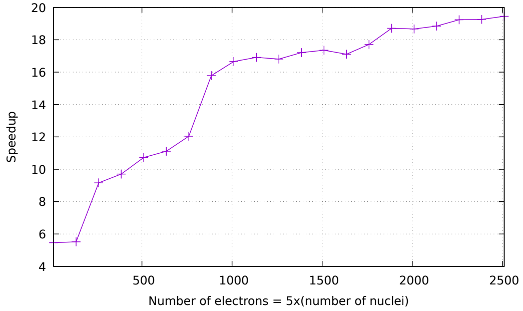
```c
#include <qmckl.h>
// ...
double            e_loc;
qmckl_context     context;

context = qmckl_context_create();

// Store WF parameters in the context
qmckl_exit_code rc = qmckl_trexio_read(context, trexio_filename, strlen(filename));
assert (rc == QMCKL_SUCCESS);

// Set the electron coordinates in the context
rc = qmckl_set_electron_coord (context, 'N', walker_num, elec_coord, walker_num*elec_num*3);
assert(rc == QMCKL_SUCCESS);

// Return the local energy at the current electron positions
rc = qmckl_get_local_energy(context, &e_loc);
// ...
```

Development strategy

1. Kernel extraction: QMC specialists agree on the mathematical expression of the problem
2. A mini-application is written to find the optimal data layout with HPC experts from real-size examples
3. The kernel is written in the documentation library
4. The documentation library is linked in a QMC code to check correctness and numerical accuracy
5. HPC experts provide an HPC version of the kernel
6. The HPC library is linked in the QMC codes of the CoE

First application : 3-body Jastrow factor

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



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

Linear algebra hot spots

GEMM,	Rank-k update,	Matrix Inversion,
GEMV,	Diagonal of GEMM,	Shermann-Morrison-Woodburry

Matrices are relatively small ($\leq 1000 \times 1000$)

- Matrices are stored in tiled format fitting a block formulation of the algorithms $\Longrightarrow$ task-based linear algebra, interleaved computation of multiple kernels
- Tile sizes will be adjusted by auto-tuning
- Increase parallelism by aggregating multiple independent walkers in matrices
- Needs fast linear algebra kernels for small matrices (tile size)
- For tiny matrices ($< 5 \times 5$) specialized versions are implemented

High-Performance strategies

Tuning

- Optimization is guided by analysis with MAQAO[a].
- Specialized versions of critical hot-spots
- Monitoring of the use of the library to choose most efficient versions
- Optimizations guided by monitoring numerical accuracy (Verificarlo[b])

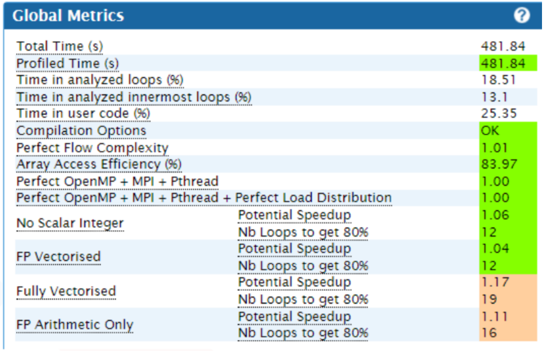
[a] https://maqao.org
[b] https://github.com/verificarlo/verificarlo

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

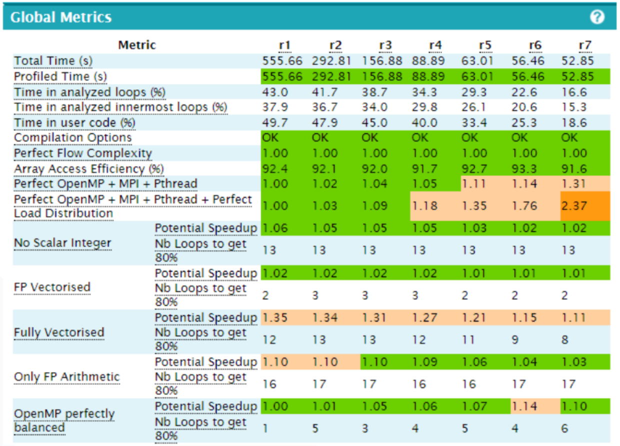
Total Time (s)		481.84
Profiled Time (s)		481.84
Time in analyzed loops (%)		18.51
Time in analyzed innermost loops (%)		13.1
Time in user code (%)		25.35
Compilation Options		OK
Perfect Flow Complexity		1.01
Array Access Efficiency (%)		83.97
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + 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

Summary

Summary

- QMC codes integrated in an ecosystem of multiple codes for high-accuracy quantum chemistry
- Development of open-source libraries to be used in the TREX codes and beyond
- Libraries focus on *performance*, *portability* and *productivity*
- Strategies to make the collaboration between physicists/chemists and HPC experts optimal

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`

Bonus slides

Numerical analysis with Verificarlo

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

- **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

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

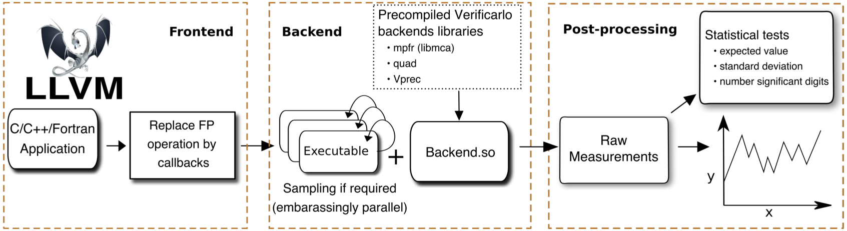
[1] C. Denis *et al.* doi:10.1109/ARITH.2016.31
[2] Y Chatelain *et al.* doi:10.1007/978-3-030-29400-7_34

The Verificarlo pipeline

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

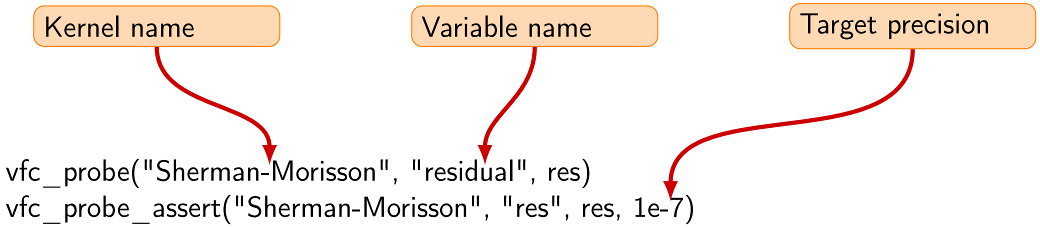
$$z = fl[x + y] = (x + y)(1 + \delta)$$

- 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


LLVM
Frontend
C/C++/Fortran Application
Replace FP operation by callbacks
Backend
Precompiled Verificarlo backends libraries
• mpfr (libmca)
• quad
• Vprec
Executable
+
Backend.so
Sampling if required (embarassingly parallel)
Post-processing
Statistical tests
• expected value
• standard deviation
• number significant digits
Raw Measurements
y
x

Continuous-Integration precision tracking

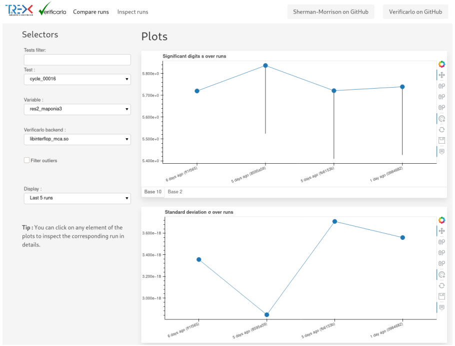
- Each push to QMCkl triggers a Verificarlo analysis.
- QMCkl kernels unit tests are augmented with probes:
  - track a scalar value precision
  - ensure that a target precision is reached



```
vfc_probe("Sherman-Morisson", "residual", res)
vfc_probe_assert("Sherman-Morisson", "res", res, 1e-7)
```

Verificarlo CI

Compare runs



Selectors

Tests filter:

Test : cycle_00016

Variable : res2_maponia3

Verificarlo backend : libinterflop_mca.so

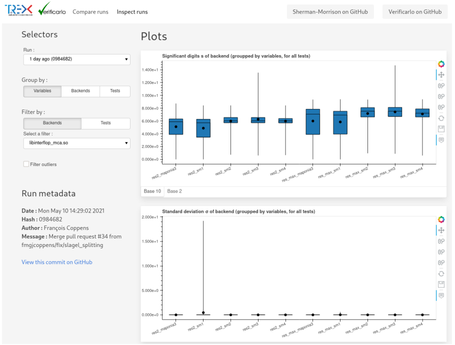
☐ Filter outliers

Display : Last 5 runs

Tip : You can click on any element of the plots to inspect the corresponding run in details.

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

Inspect runs



Selectors

Run : 1 day ago (0984682)

Group by : [✓ Variables] Backends Tests

Filter by : [✓ Backends] Tests

Select a filter : libinterflop_mca.so

☐ Filter outliers

Run metadata

Date : Mon May 10 14:29:02 2021
Hash : 0984682
Author : François Coppens
Message : Merge pull request #34 from fmgjcoppens/fix/slagel_splitting

View this commit on GitHub

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

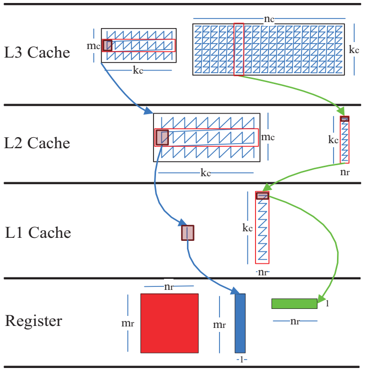
Example: Specialized DGEMM kernel

Simple algorithm

- Simple micro kernel (GotoDGEMM[a])
- Code written using `asm` to force good code generation by compilers
- Tiling scheme[b]

[a] doi:10.1145/1356052.1356053
[b] doi:10.1109/ICPP.2015.29

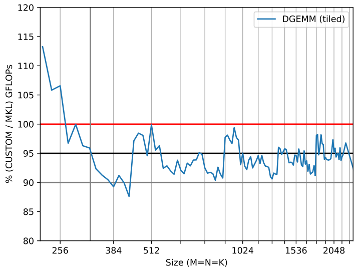
Tiling scheme



Example: Specialized DGEMM kernel

Benchmarks

- Comparison of MKL vs Specialized DGEMM



- Strong impact on MKL performance due to the number of consecutive executions
- Favorable comparison for MKL: Many consecutive executions to amortize setup cost, JIT, Skylake CPU

Why provide our own DGEMM?

- Decent performance (within 10% of MKL) guaranteed independently of the compiler and BLAS variant
- Simple code (a few lines of code)
- Open source : can be modified easily
- Can be rewritten in different languages to increase portability (MIPP[2])
- Allows to keep control on parallelism
- Makes autotuning simple

[2]https://github.com/aff3ct/MIPP