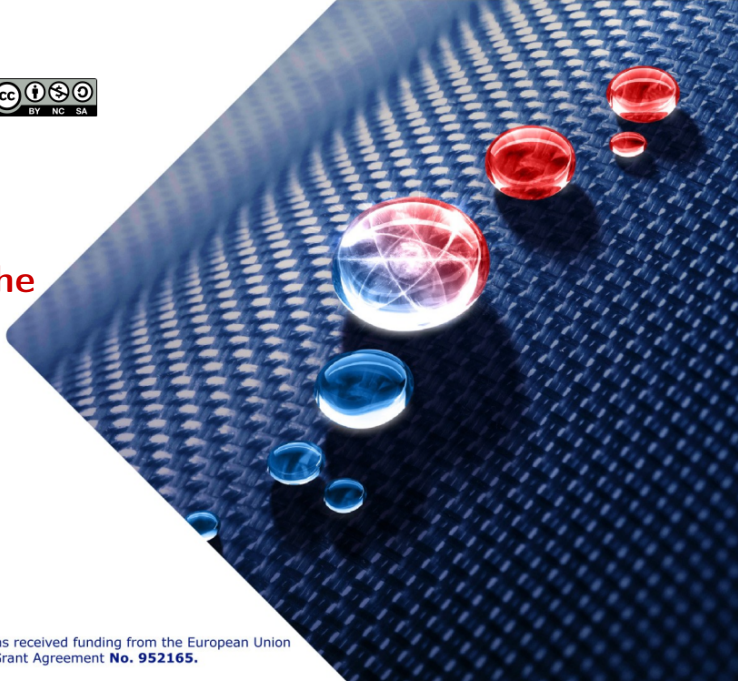
Targeting Real chemical accuracy at the EXascale

Libraries developed in the TREX CoE

A. Scemama, V.G. Chilkuri

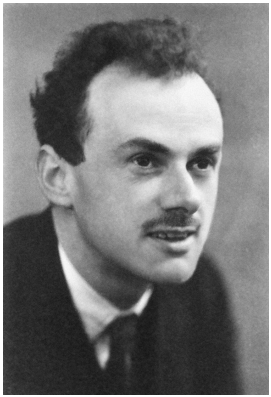
20/10/2021

University of Toulouse/CNRS, LCPQ (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.**

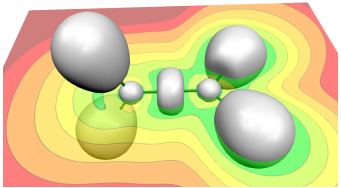
The TREX CoE

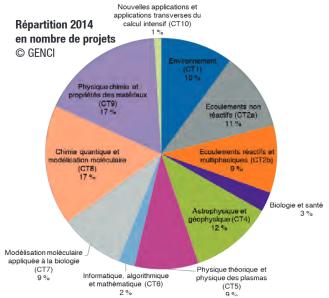


"The underlying physical laws necessary for the mathematical theory of a large part of physics and the whole of chemistry are thus completely known, and the difficulty is only that the exact application of these laws leads to equations much too complicated to be soluble. It therefore becomes desirable that approximate practical methods of applying quantum mechanics should be developed, which can lead to an explanation of the main features of complex atomic systems without too much computation."

Paul Dirac (1926) *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character, Vol. 123, No. 792* (6 April 1929)

- Description of matter with quantum mechanics (Schrödinger equation)
- Users: Theoretical chemists / Physicists





Applications

- Health	Drug design
- Electronics	Nano- and micro-electronics
- Materials	Carbon Nanotubes, graphene
- Catalysis	Enzymatic reations, 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 CoE: Targeting REal chemical accuracy at the eXascale
- Started in Oct. 2020
- Objective: Make codes ready for exascale systems

- TREX CoE: Targeting REal chemical accuracy at the eXascale
- Started in Oct. 2020
- Objective: Make codes ready for exascale systems
- Two regimes for exascale:
  - Single exascale run
  - Thousands of petascale simulations in high-throughput (HTC)

- TREX CoE: Targeting REal chemical accuracy at the eXascale
- Started in Oct. 2020
- Objective: Make codes ready for exascale systems
- Two regimes for exascale:
  - Single exascale run
  - Thousands of petascale simulations in high-throughput (HTC)
- How: Instead of re-writing codes, provide libraries
  - One library for high-performance (QMCkl)
  - One library for exchanging information between codes (TREXIO)

QMC kernel library (QMCkl)

Problem: Stochastic resolution of the Schrödinger equation for N electrons

$$
\begin{aligned}
E &= \frac{\int \mathrm{d}r_1 \dots \mathrm{d}r_N \, \Phi(r_1, \dots, r_N) \mathcal{H} \Phi(r_1, \dots, r_N)}{\int \mathrm{d}r_1 \dots \mathrm{d}r_N \, \Phi(r_1, \dots, r_N) \Phi(r_1, \dots, r_N)} \\
&\sim \sum \frac{\mathcal{H}\Psi(r_1, \dots, r_N)}{\Psi(r_1, \dots, r_N)}, \text{ sampled with } (\Psi \times \Phi)
\end{aligned}
$$

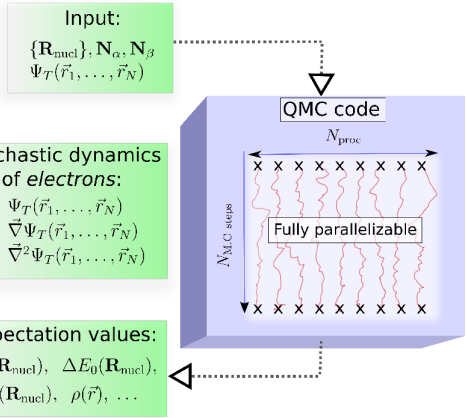
- $\mathcal{H}$: Hamiltonian operator
- E: Energy
- $r_1, \dots, r_N$: Electron coordinates
- Φ: Almost exact wave function
- Ψ: Trial wave function

Good

- Very low memory requirements (no integrals)
- Distribute walkers on different cores or compute nodes
- No blocking communication: near-ideal scaling
- One Monte Carlo step is fast ($\sim 1ms$)

Bad

- Difficulty to parallelize within a QMC trajectory: depends on the number of electrons
- Linear algebra with small matrices



- Progress in quantum chemistry may require codes with new ideas/algorithms
- New ideas/algorithms are implemented by physicists/chemists
- Different scientists have different programming language knowledge/preference
- Exascale machines will be horribly complex to program

- Progress in quantum chemistry may require codes with new ideas/algorithms
- New ideas/algorithms are implemented by physicists/chemists
- Different scientists have different programming language knowledge/preference
- Exascale machines will be horribly complex to program

Question

Is it reasonable to ask physicists/chemists to write codes for exascale machines?

$$Z^{(n+1)} = Z^{(n)} + aX + Y$$

```
1    do i=1,n
2      Z(i) = Z(i) + A * X(i) + Y(i)
3    end do
```

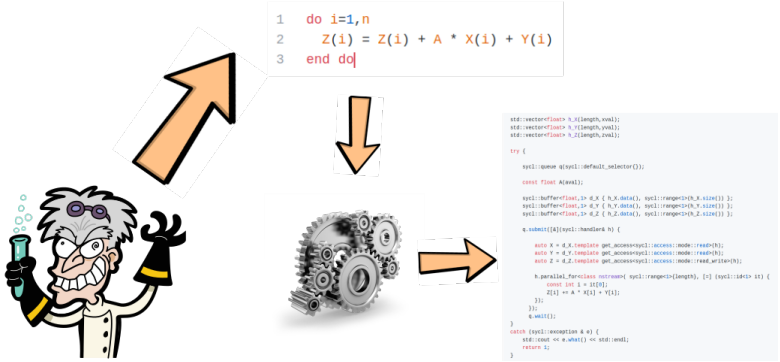
(from https://github.com/jeffhammond/dpcpp-tutorial)

```
std::vector<float> h_X(length,xval);
std::vector<float> h_Y(length,yval);
std::vector<float> h_Z(length,zval);

try {

    sycl::queue q(sycl::default_selector{});

    const float A(aval);

    sycl::buffer<float,1> d_X { h_X.data(), sycl::range<1>(h_X.size()) };
    sycl::buffer<float,1> d_Y { h_Y.data(), sycl::range<1>(h_Y.size()) };
    sycl::buffer<float,1> d_Z { h_Z.data(), sycl::range<1>(h_Z.size()) };

    q.submit([&](sycl::handler& h) {

        auto X = d_X.template get_access<sycl::access::mode::read>(h);
        auto Y = d_Y.template get_access<sycl::access::mode::read>(h);
        auto Z = d_Z.template get_access<sycl::access::mode::read_write>(h);

        h.parallel_for<class nstream>( sycl::range<1>{length}, [=] (sycl::id<1> it) {
            const int i = it[0];
            Z[i] += A * X[i] + Y[i];
        });
      });
      q.wait();
}
catch (sycl::exception & e) {
    std::cout << e.what() << std::endl;
    return 1;
}
```

```
std::vector<float> h_X(length,xval);
std::vector<float> h_Y(length,yval);
std::vector<float> h_Z(length,zval);

try {

    sycl::queue q(sycl::default_selector{});

    const float A(aval);

    sycl::buffer<float,1> d_X { h_X.data(), sycl::range<1>(h_X.size()) };
    sycl::buffer<float,1> d_Y { h_Y.data(), sycl::range<1>(h_Y.size()) };
    sycl::buffer<float,1> d_Z { h_Z.data(), sycl::range<1>(h_Z.size()) };

    q.submit([&](sycl::handler& h) {

        auto X = d_X.template get_access<sycl::access::mode::read>(h);
        auto Y = d_Y.template get_access<sycl::access::mode::read>(h);
        auto Z = d_Z.template get_access<sycl::access::mode::read_write>(h);

        h.parallel_for<class nstream>( sycl::range<1>{length}, [=] (sycl::id<1> it) {
            const int i = it[0];
            Z[i] += A * X[i] + Y[i];
        });
      });
      q.wait();
}
catch (sycl::exception & e) {
    std::cout << e.what() << std::endl;
    return 1;
}
```

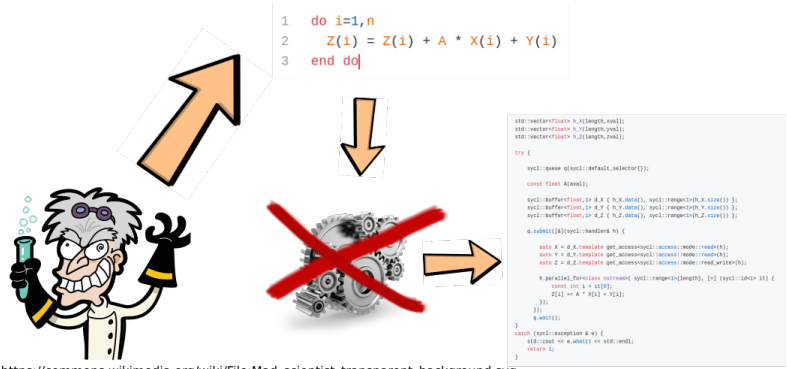
https://commons.wikimedia.org/wiki/File:Mad_scientist_transparent_background.svg

A compiler[1] that can read an average researcher's code and transform it into highly efficient code on an exascale machine.



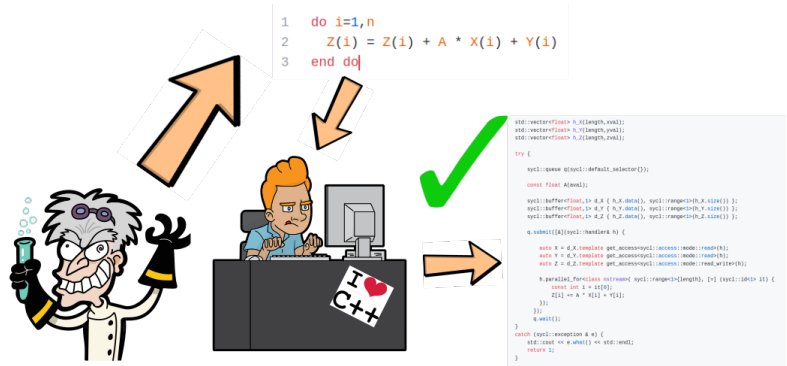
[1]Wikipedia: A compiler is a computer program that translates computer code written in one programming language (the source language) into another language (the target language)

Artificial Intelligence is not ready yet . . .



https://commons.wikimedia.org/wiki/File:Mad_scientist_transparent_background.svg

. . . so let's use *Natural Intelligence* and add a human layer between the machine and the researchers : a bio-compiler



https://commons.wikimedia.org/wiki/File:Mad_scientist_transparent_background.svg

- Identify the common computational kernels of QMC
- Implement these kernels in a human-readable library (QMC experts)
- *Bio-compile* the human-readable library in a HPC-library (HPC experts)
- Scientists can link either library with their codes

For scientists

- We don't impose a programming language
- The code can stay easy to understand by the physicists/chemists
  Performance-related aspects are delegated to the library
- Codes will not die with a change in architecture
- Scientific code development does not break the performance
- Scientists don't lose control on their codes

Benefits of this model

For scientists

- We don't impose a programming language
- The code can stay easy to understand by the physicists/chemists
  Performance-related aspects are delegated to the library
- Codes will not die with a change in architecture
- Scientific code development does not break the performance
- Scientists don't lose control on their codes

Separation of concerns

- Scientists will never have to manipulate low-level HPC code
- HPC experts will not be required to be experts in theoretical physics
- Better re-use of the optimization effort among the community

- The API is C-compatible: QMCkl appears like a C library $\Longrightarrow$ can be used in all other languages
- System functions in C (memory allocation, thread safety, *etc*)
- Computational kernels in Fortran for readability
- A lot of documentation (remember: the HPC compiler is a human!)

Literate programming is a programming paradigm introduced by Donald Knuth in which a computer program is given an explanation of its logic in a natural language, such as English, interspersed with snippets of macros and traditional source code, from which compilable source code can be generated. (Wikipedia)

Literate programming with *org-mode*:

- Here, comments are more important than code
- Can add graphics, LaTeXformulas, tables, etc
- Documentation always synchronized with the code
- Some functions can be generated by embedded scripts
- Web site auto-generated when code is pushed

Instead of writing comments documenting code, we write code illustrating documentation.

File Edit Options Buffers Tools Table Org Text Help

Save Undo

#+TITLE: **Atomic Orbitals**
#+SETUPFILE: ../docs/theme.setup
#+INCLUDE: ../tools/lib.org

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_l. The radial function is a linear combination of \emph{primitive} functions that can be of type Slater ($p=1$) or Gaussian ($p=2$):

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

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 to unity. As this normalization requires the ability to compute overlap integrals, it should be written in the file to ensure that the file is self-contained and does not require the client program to have the ability to compute such integrals.

Atomic orbitals (AOs) are defined as

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

where $\theta(i)$ returns the shell on which the AO is expanded, and $\eta(i)$ denotes which angular function is chosen.

In this section we describe the kernels used to compute the values, gradients and Laplacian of the atomic basis functions.

- **Headers** :noexport:
- **Context...**
- **Polynomial part...**
- **Radial part**
  - **Gaussian basis functions**

~qmckl_ao_gaussian_vgl~ computes the values, gradients and Laplacians at a given point of ~-n-~ Gaussian functions centered at the same point:

$$v_i = \exp(-\alpha_i |X-R|^2)$$

U:@--- **qmckl_ao.org** Top (1459,0) <N> Git:context (Org ARev ² Undo-Tree Fill) Mail [1]

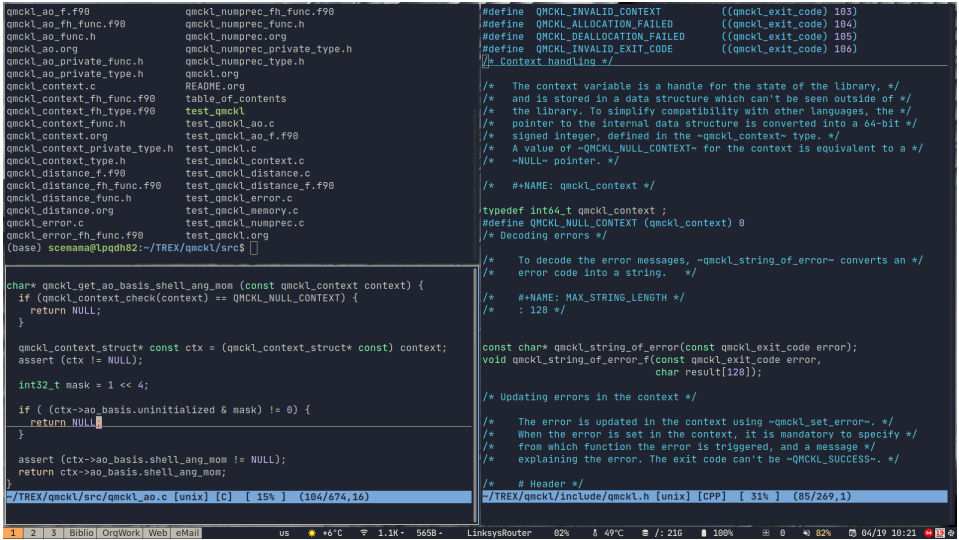
~-context-~	input	Global state
~-X(3)-~	input	Array containing the coordinates of the points
~-R(3)-~	input	Array containing the x,y,z coordinates of the center
~-n-~	input	Number of computed Gaussians
~-A(n)-~	input	Exponents of the Gaussians
~-VGL(ldv,5)-~	output	Value, gradients and Laplacian of the Gaussians
~-ldv-~	input	Leading dimension of array ~-VGL-~

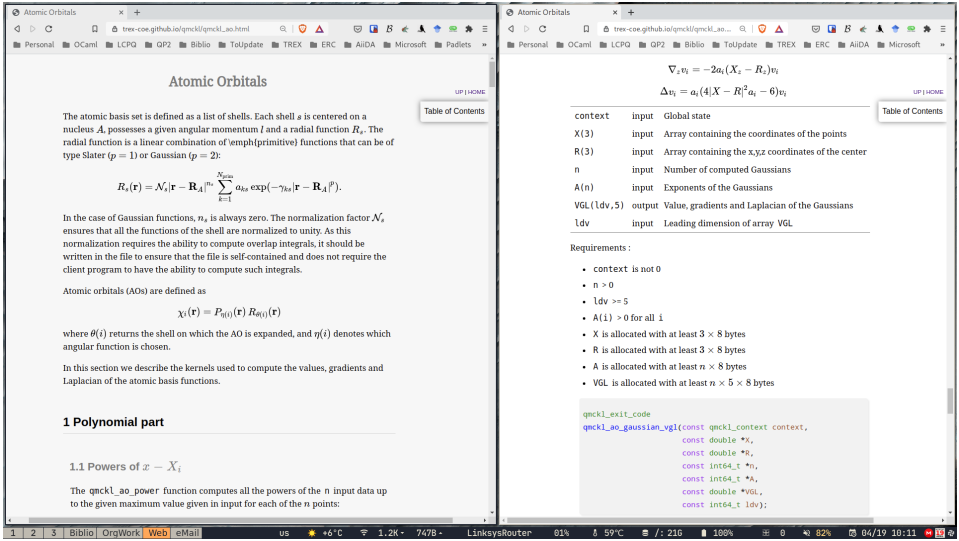
Requirements :

- ~-context-~ is not 0
- ~-n-~ > 0
- ~-ldv-~ >= 5
- ~-A(i)-~ > 0 for all ~-i-~
- ~-X-~ is allocated with at least 3×8 bytes
- ~-R-~ is allocated with at least 3×8 bytes
- ~-A-~ is allocated with at least $n \times 8$ bytes
- ~-VGL-~ is allocated with at least $n \times 5 \times 8$ bytes

```
#+begin_src c :tangle (eval h_func)
qmckl_exit_code
qmckl_ao_gaussian_vgl(const qmckl_context context,
                      const double *X,
                      const double *R,
                      const int64_t *n,
                      const int64_t *A,
                      const double *VGL,
                      const int64_t ldv);
#+end_src

#+begin_src f90 :tangle (eval f)
integer function qmckl_ao_gaussian_vgl_f(context, X, R, n, A, VGL, ldv) result(info)
  use qmckl
  implicit none
  integer*8 , intent(in)  :: context
  real*8    , intent(in)  :: X(3), R(3)
  integer*8 , intent(in)  :: n
  real*8    , intent(in)  :: A(n)
  real*8    , intent(out) :: VGL(ldv,5)
  integer*8 , intent(in)  :: ldv

  integer*8               :: i,j
  real*8                  :: Y(3), r2, t, u, v
```

U:@--- **qmckl_ao.org** 85% (1493,0) <N> Git:context (Org ARev ² Undo-Tree Fill) Mail [1]


qmckl_ao_f.f90 qmckl_numprec_fh_func.f90
qmckl_ao_fh_func.f90 qmckl_numprec_func.h
qmckl_ao_func.h qmckl_numprec.org
qmckl_ao.org qmckl_numprec_private_type.h
qmckl_ao_private_func.h qmckl_numprec_type.h
qmckl_ao_private_type.h qmckl.org
qmckl_context.c README.org
qmckl_context_fh_func.f90 table_of_contents
qmckl_context_fh_type.f90 test_qmckl
qmckl_context_func.h test_qmckl_ao.c
qmckl_context.org test_qmckl_ao_f.f90
qmckl_context_private_type.h test_qmckl.c
qmckl_context_type.h test_qmckl_context.c
qmckl_distance_f.f90 test_qmckl_distance.c
qmckl_distance_fh_func.f90 test_qmckl_distance_f.f90
qmckl_distance_func.h test_qmckl_error.c
qmckl_distance.org test_qmckl_memory.c
qmckl_error.c test_qmckl_numprec.c
qmckl_error_fh_func.f90 test_qmckl.org
(base) scemama@lpqdh82:~/TREX/qmckl/src$
char* qmckl_get_ao_basis_shell_ang_mom (const qmckl_context context) {
if (qmckl_context_check(context) == QMCKL_NULL_CONTEXT) {
return NULL;
}
qmckl_context_struct* const ctx = (qmckl_context_struct* const) context;
assert (ctx != NULL);
int32_t mask = 1 << 4;
if ((ctx->ao_basis.uninitialized & mask) != 0) {
return NULL;
}
assert (ctx->ao_basis.shell_ang_mom != NULL);
return ctx->ao_basis.shell_ang_mom;
}
~/TREX/qmckl/src/qmckl_ao.c [unix] [C] [15%] (104/674,16)
#define QMCKL_INVALID_CONTEXT ((qmckl_exit_code) 103)
#define QMCKL_ALLOCATION_FAILED ((qmckl_exit_code) 104)
#define QMCKL_DEALLOCATION_FAILED ((qmckl_exit_code) 105)
#define QMCKL_INVALID_EXIT_CODE ((qmckl_exit_code) 106)
/* Context handling */
/* The context variable is a handle for the state of the library, */
/* and is stored in a data structure which can't be seen outside of */
/* the library. To simplify compatibility with other languages, the */
/* pointer to the internal data structure is converted into a 64-bit */
/* signed integer, defined in the ~qmckl_context~ type. */
/* A value of ~QMCKL_NULL_CONTEXT~ for the context is equivalent to a */
/* ~NULL~ pointer. */
/* #+NAME: qmckl_context */
typedef int64_t qmckl_context ;
#define QMCKL_NULL_CONTEXT (qmckl_context) 0
/* Decoding errors */
/* To decode the error messages, ~qmckl_string_of_error~ converts an */
/* error code into a string. */
/* #+NAME: MAX_STRING_LENGTH */
/* : 128 */
const char* qmckl_string_of_error(const qmckl_exit_code error);
void qmckl_string_of_error_f(const qmckl_exit_code error,
char result[128]);
/* Updating errors in the context */
/* The error is updated in the context using ~qmckl_set_error~. */
/* When the error is set in the context, it is mandatory to specify */
/* from which function the error is triggered, and a message */
/* explaining the error. The exit code can't be ~QMCKL_SUCCESS~. */
/* # Header */
~/TREX/qmckl/include/qmckl.h [unix] [CPP] [31%] (85/269,1)
1 2 3 Biblio OrgWork Web eMail us +6°C 1.1K- 565B- LinksysRouter 82% 49°C /: 21G 100% 0 82% 04/19 10:21


Atomic Orbitals
trex-coe.github.io/qmckl/qmckl_ao.html
Personal OCaml LCPQ QP2 Biblio ToUpdate TREX ERC AiiDA Microsoft Padlets
Atomic Orbitals
UP | HOME
Table of Contents
The atomic basis set is defined as a list of shells. Each shell s is centered on a nucleus A_s possesses a given angular momentum l and a radial function R_s. The radial function is a linear combination of \emph{primitive} functions that can be of type Slater ($p = 1$) or Gaussian ($p = 2$):
$$R_s(\mathbf{r}) = \mathcal{N}_s |\mathbf{r} - \mathbf{R}_A|^{n_s} \sum_{k=1}^{N_{prim}} a_{ks} \exp(-\gamma_{ks} |\mathbf{r} - \mathbf{R}_A|^p).$$
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 to unity. As this normalization requires the ability to compute overlap integrals, it should be written in the file to ensure that the file is self-contained and does not require the client program to have the ability to compute such integrals.
Atomic orbitals (AOs) are defined as
$$\chi_i(\mathbf{r}) = P_{\eta(i)}(\mathbf{r})\, R_{\theta(i)}(\mathbf{r})$$
where $\theta(i)$ returns the shell on which the AO is expanded, and $\eta(i)$ denotes which angular function is chosen.
In this section we describe the kernels used to compute the values, gradients and Laplacian of the atomic basis functions.
1 Polynomial part
1.1 Powers of $x - X_i$
The qmckl_ao_power function computes all the powers of the n input data up to the given maximum value given in input for each of the n points:
Atomic Orbitals
UP | HOME
Table of Contents
$$\nabla_z v_i = -2a_i(X_z - R_z)v_i$$
$$\Delta v_i = a_i(4|X - R|^2 a_i - 6)v_i$$
context input Global state
X(3) input Array containing the coordinates of the points
R(3) input Array containing the x,y,z coordinates of the center
n input Number of computed Gaussians
A(n) input Exponents of the Gaussians
VGL(ldv,5) output Value, gradients and Laplacian of the Gaussians
ldv input Leading dimension of array VGL
Requirements :
context is not 0
n > 0
ldv >= 5
A(i) > 0 for all i
X is allocated with at least 3×8 bytes
R is allocated with at least 3×8 bytes
A is allocated with at least $n \times 8$ bytes
VGL is allocated with at least $n \times 5 \times 8$ bytes
qmckl_exit_code
qmckl_ao_gaussian_vgl(const qmckl_context context,
const double *X,
const double *R,
const int64_t *n,
const int64_t *A,
const double *VGL,
const int64_t ldv);
1 2 3 Biblio OrgWork Web eMail us +6°C 1.2K- 747B- LinksysRouter 81% 59°C /: 21G 100% 0 82% 04/19 10:11

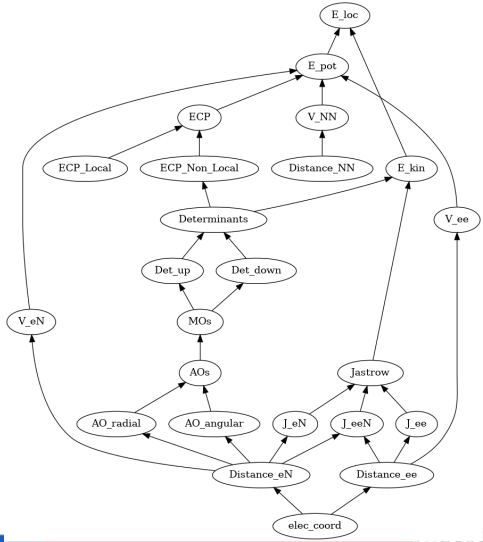
At each QMC step, we need to evaluate $E_{\text{loc}}(r_1, \ldots, r_N) = \frac{\hat{H}\Psi(r_1,\ldots,r_N)}{\Psi(r_1,\ldots,r_N)}$:

- $\Psi(r_1, \ldots, r_N)$
- $\Delta_i \Psi(r_1, \ldots, r_i, \ldots, r_N)$: kinetic energy
- $\vec{\nabla}_i \Psi(r_1, \ldots, r_i, \ldots, r_N)$: drift in the stochastic process

Main kernels

- AOs: $\chi(r), \vec{\nabla}\chi(r), \Delta\chi(r)$
- MOs: $\phi(r), \vec{\nabla}\phi(r), \Delta\phi(r)$
- Slater determinants (value, gradient, Laplacian)
- Pseudo-potential
- Jastrow correlation factor (eN, ee, eeN)

$$
\begin{aligned}
E_{\text{loc}}(\text{R}) &= E_{\text{pot}}(\text{R}) + E_{\text{kin}}(\text{R}) \\
E_{\text{pot}}(\text{R}) &= V_{ee}(\text{R}) + V_{eN}(\text{R}) + V_{NN}(\text{R}) + \text{ECP}(\text{R}) \\
E_{\text{kin}}(\text{R}) &= -\frac{1}{2}\frac{\Delta\Psi(\text{R})}{\Psi(\text{R})} \\
\Psi(\text{R}) &= \Phi(\text{R})J(\text{R}) \\
&\dots
\end{aligned}
$$



1. Kernel extraction: QMC exterts agree on the mathematical expression of the problem
2. A mini-application is written to find the best data layout with HPC experts from real-size examples
3. The kernel is written in the documentation library
4. HPC experts provide an HPC version of the kernel with the same API
5. The library is linked in the QMC codes of the CoE

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

can be rewritten as

$$J_{\text{een}}(\mathrm{r},\mathrm{R}) = \sum_{p=2}^{N_{\text{nord}}} \sum_{k=0}^{p-1} \sum_{l=0}^{p-k-2\delta_{k,0}} \sum_{\alpha=1}^{N_{\text{nucl}}} c_{lkp\alpha} \sum_{i=1}^{N_{\text{elec}}} \bar{\mathrm{R}}_{i,\alpha,(p-k-l)/2} \, \bar{\mathrm{P}}_{i,\alpha,k,(p-k+l)/2} \text{ (↓ complexity)}$$

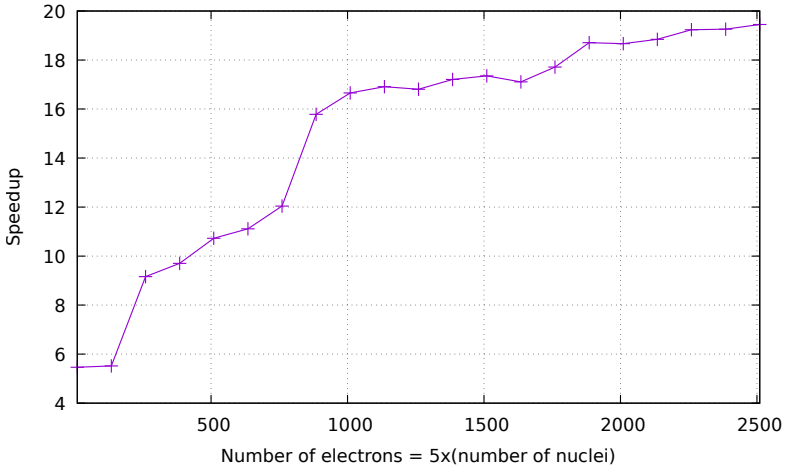
with

$$\bar{\mathrm{P}}_{i,\alpha,k,l} = \sum_{j=1}^{N_{\text{elec}}} \bar{\mathrm{r}}_{i,j,k} \, \bar{\mathrm{R}}_{j,\alpha,l}. \text{ (GEMM)}$$

$$\nabla_{im} J_{\text{een}}(\mathrm{r},\mathrm{R}) = \sum_{p=2}^{N_{\text{nord}}} \sum_{k=0}^{p-1} \sum_{l=0}^{p-k-2\delta_{k,0}} \sum_{\alpha=1}^{N_{\text{nucl}}} c_{lkp\alpha} \sum_{i=1}^{N_{\text{elec}}} \bar{\mathrm{G}}_{i,m,\alpha,(p-k-l)/2}\, \bar{\mathrm{P}}_{i,\alpha,k,(p-k+l)/2} +$$
$$\bar{\mathrm{G}}_{i,m,\alpha,(p-k+l)/2}\, \bar{\mathrm{P}}_{i,\alpha,k,(p-k-l)/2} + \bar{\mathrm{R}}_{i,\alpha,(p-k-l)/2}\, \bar{\mathrm{Q}}_{i,m,\alpha,k,(p-k+l)/2} +$$
$$\bar{\mathrm{R}}_{i,\alpha,(p-k+l)/2}\, \bar{\mathrm{Q}}_{i,m,\alpha,k,(p-k-l)/2} + \delta_{m,4}\Big($$
$$\bar{\mathrm{G}}_{i,1,\alpha,(p-k+l)/2}\, \bar{\mathrm{Q}}_{i,1,\alpha,k,(p-k-l)/2} + \bar{\mathrm{G}}_{i,2,\alpha,(p-k+l)/2}\, \bar{\mathrm{Q}}_{i,2,\alpha,k,(p-k-l)/2} +$$
$$\bar{\mathrm{G}}_{i,3,\alpha,(p-k+l)/2}\, \bar{\mathrm{Q}}_{i,3,\alpha,k,(p-k-l)/2} + \bar{\mathrm{G}}_{i,1,\alpha,(p-k-l)/2}\, \bar{\mathrm{Q}}_{i,1,\alpha,k,(p-k+l)/2} +$$
$$\bar{\mathrm{G}}_{i,2,\alpha,(p-k-l)/2}\, \bar{\mathrm{Q}}_{i,2,\alpha,k,(p-k+l)/2} + \bar{\mathrm{G}}_{i,3,\alpha,(p-k-l)/2}\, \bar{\mathrm{Q}}_{i,3,\alpha,k,(p-k+l)/2}\Big)$$

with

$$\bar{\mathrm{G}}_{i,m,\alpha,l} = \frac{\partial \left(R_{i\alpha}\right)^l}{\partial r_i}, \qquad \bar{\mathrm{g}}_{i,m,j,k} = \frac{\partial \left(r_{ij}\right)^k}{\partial r_i}, \qquad \text{and } \bar{\mathrm{Q}}_{i,m,\alpha,k,l} = \sum_{j=1}^{N_{\text{elec}}} \bar{\mathrm{g}}_{i,m,j,k}\, \bar{\mathrm{R}}_{j,\alpha,l}$$



$\sim$ 80% of the AVX-512 peak is reached on a Skylake CPU with MKL.

- A few QMC trajectories per compute node $\Longrightarrow$ distributed parallelism not necessary inside the library. Delegated to the calling program.

- A few QMC trajectories per compute node $\Longrightarrow$ distributed parallelism not necessary inside the library. Delegated to the calling program.
- Agregate multiple trajectories to increase the size of the matrices and give enough to eat to GPUs

- A few QMC trajectories per compute node $\Longrightarrow$ distributed parallelism not necessary inside the library. Delegated to the calling program.
- Agregate multiple trajectories to increase the size of the matrices and give enough to eat to GPUs
- For each kernel, develop both a CPU and a GPU variant (OpenMP)
- Let StarPU schedule kernels

- A few QMC trajectories per compute node $\Longrightarrow$ distributed parallelism not necessary inside the library. Delegated to the calling program.
- Agregate multiple trajectories to increase the size of the matrices and give enough to eat to GPUs
- For each kernel, develop both a CPU and a GPU variant (OpenMP)
- Let StarPU schedule kernels
- Use internally matrices in tiled format for linear algebra (tiled LA) to:
  - Keep control on the performance
  - Schedule tiled LA tasks together with other kernels
  - Enable hybrid parallelism for large LA kernels
  - Facilitate usage of GPUs for matrices too large to fit on the GPU

See Vijay's slides

Summary

- MIPP for fast and portable matrix multiplication (and more) among tiles
- Advice for efficient use of StarPU:
  - Merging many small tasks into one big?
- Usage of Chameleon for tiled LA with StarPU?
- ...