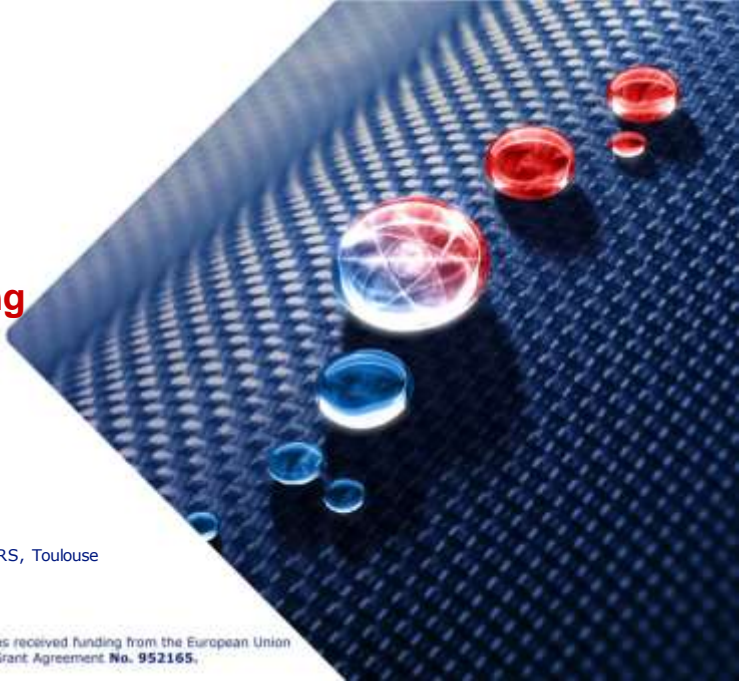


QMCKI: A Unified Approach to Accelerating Quantum Monte Carlo Codes

Anthony Scemama

5/02/2024

Lab. Chimie et Physique Quantiques, FERMI, UPS/CNRS, Toulouse
(France)





The TREX European Center of
Excellence



Partners

UNIVERSITY OF TWENTE



JÜLICH
Forschungszentrum



MEGWARE



STU
SLOVAK UNIVERSITY OF
TECHNOLOGY IN BRATISLAVA

Trust-IT Services
Communicating IT in motion

universität
wien

UVSQ
UNIVERSITÉ PARIS-SACLAY

Codes

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

CHAMP (Claudia Filippi)

- Wave function optimization:
Jastrow, CI, MOs
- Ground/Excited states
- Geometry optimization

CHAMP (Claudia Filippi)

- Wave function optimization:
Jastrow, CI, MOs
- Ground/Excited states
- Geometry optimization

TurboRVB (Sandro Sorella + Michele Casula)

- Molecular and Periodic systems
- JAGP, Pfaffian, ...
- LRDMC

CHAMP (Claudia Filippi)

- Wave function optimization: Jastrow, CI, MOs
- Ground/Excited states
- Geometry optimization

TurboRVB (Sandro Sorella + Michele Casula)

- Molecular and Periodic systems
- JAGP, Pfaffian, ...
- LRDMC

QMC=Chem (Michel Caffarel + Me!)

- DMC as “Post-Full-CI” energy calculations (CIPSI)
- Very large CI expansions (millions of determinants)
- Designed with HPC in mind
- Highly optimized with W. Jalby's group (UVSQ) in 2011-2013

- 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
- How: Instead of re-writing codes, provide **libraries**
 - One library for high-performance (**QMCKI**)
 - One library for exchanging information between codes (**TREXIO**)

The QMC kernel library (QMCKI)

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

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

Question

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

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

$$Z_{n+1} = Z_n + aX_n + Y_n$$

1 do i=1, n

2 Z(i) = Z(i) + A * X(i) + Y(i)

3 end do

```

std::vector<float> h_X(length,aval);
std::vector<float> h_Y(length,aval);
std::vector<float> h_Z(length,aval);

try {

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

    const float A(aval);

    sycl::buffer<float,1> B_X { h_X.data(), sycl::range<1>(h_X.size()) };
    sycl::buffer<float,1> B_Y { h_Y.data(), sycl::range<1>(h_Y.size()) };
    sycl::buffer<float,1> B_Z { h_Z.data(), sycl::range<1>(h_Z.size()) };

    n.submit([A](sycl::handler h) {

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

        h.parallel_for<class int>(sycl::range<1>(length), [=](sycl::id<1> it) {
            const int i = it[0];
            Z[i] += A * X[i] + Y[i];
        });
    });

    h.wait();

}

catch (sycl::exception & e) {
    std::cout << e.what() << std::endl;
    return 1;
}

```



```
std::vector<float> N_x(length, 0.0);
std::vector<float> N_y(length, 0.0);
std::vector<float> N_z(length, 0.0);

try {

    spcl::queue q(spcl::default_executor());

    const float A[6][6];

    spcl::buffer<float, 2> A_X { N_x.data(), spcl::range(0)(N_x.size()) };
    spcl::buffer<float, 2> A_Y { N_y.data(), spcl::range(0)(N_y.size()) };
    spcl::buffer<float, 2> A_Z { N_z.data(), spcl::range(0)(N_z.size()) };

    q.submit([A](spcl::handler h) {

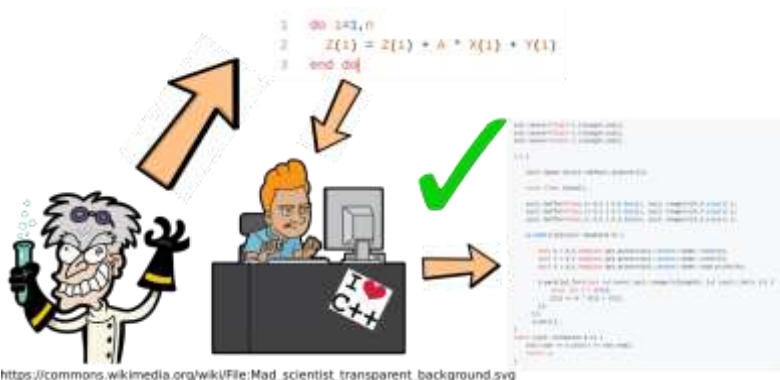
        auto X = A_X.template get_access(spcl::access::mode::read)(h);
        auto Y = A_Y.template get_access(spcl::access::mode::read)(h);
        auto Z = A_Z.template get_access(spcl::access::mode::read_write)(h);

        h.parallel_for_each_out_range(spcl::range(0)(length), [=] (spcl::id<int> id) {
            auto int id = id[0];
            Z[id] += A[0][id] + Y[id];
        });
    });
    q.wait();
}
catch (spcl::exception & e) {
    std::cout << e.what() << "\n";
    return 1;
}
```

A compiler¹ that can read an average researcher's code and transform it into highly efficient code on an exascale machine.

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

... so we decided to use *Natural Intelligence*, and add a human layer between the machine and the researchers : a **biological compiler**



- 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

- The choice of the programming language is not imposed to the scientist
- 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 hardware
- Scientific code development does not break the performance
- Scientists don't lose control on their codes

For scientists

- The choice of the programming language is not imposed to the scientist
- 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 hardware
- 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 QMCKI Documentation
library

- The API is C-compatible: QMCKI appears to scientists like a C library \Rightarrow can be used in all other languages
- System functions in programmed C (memory allocation, thread safety, *etc*)
- Computational kernels programmed in simple 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, L^AT_EX formulas, 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.

Atomic Orbitals

The atomic basis set is defined as a list of shells. Each shell is represented as a radius A , possessing a given angular momentum l and a radial function R_l . The radial function is a linear combination of (spherical) functions that can be of type $\exp(-\lambda r)$ or $\exp(-\lambda r - \beta/r)$:

$$R_l(r) = N_l r^{-\beta} \sum_{i=1}^n a_i \exp(-\lambda_i r - \beta/r)$$

In the case of Gaussian functions, β is always zero. The normalization factor N_l means that all the functions of the shell are normalized to unity. As this normalization requires the ability to compute nesting 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

$$\chi_i(\mathbf{r}) = R_{l_i}(r) Y_{l_i m_i}(\theta, \phi)$$

where $R_{l_i}(r)$ denotes the shell on which the AO is expanded, and $Y_{l_i m_i}(\theta, \phi)$ denotes which angular function it denotes.

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

1 Polynomial part

1.1 Powers of $x - \beta$

The `spec1_at_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:

$$\nabla^2 \psi_0 = -\beta \psi_0 (X - \beta)^{-2}$$

$$\Delta \psi_0 = \alpha_1 (4N - \beta^2) \psi_0 - 6\beta \psi_0$$

<code>radius1</code>	input	Global radii
<code>l1</code>	input	Array containing the radii indices of the points
<code>l2</code>	input	Array containing the l type coordinates of the points
<code>n</code>	input	Number of computed functions
<code>A(i)</code>	input	Eigenvalues of the Gaussian
<code>VD,CDV,DD</code>	output	Values, gradients and Laplacians of the Gaussian
<code>lbr</code>	input	Leading dimension of array <code>VD</code>

Implementation:

- `radius1` is real k
- `n` is > 0
- `lbr` is 1
- `A(i)` is k for all i
- `X` is allocated with at least 3×8 bytes
- `D` is allocated with at least 3×8 bytes
- `A` is allocated with at least $n \times 8$ bytes
- `VD` is allocated with at least $n \times 3 \times 8$ bytes

```
spec1_at_power
spec1_at_power(n, radius1, ngl) {
    int i, j, k, l, m, n, ngl, nmax, nmin, ntot, nval, nval2, nval3, nval4, nval5, nval6, nval7, nval8, nval9, nval10, nval11, nval12, nval13, nval14, nval15, nval16, nval17, nval18, nval19, nval20, nval21, nval22, nval23, nval24, nval25, nval26, nval27, nval28, nval29, nval30, nval31, nval32, nval33, nval34, nval35, nval36, nval37, nval38, nval39, nval40, nval41, nval42, nval43, nval44, nval45, nval46, nval47, nval48, nval49, nval50, nval51, nval52, nval53, nval54, nval55, nval56, nval57, nval58, nval59, nval60, nval61, nval62, nval63, nval64, nval65, nval66, nval67, nval68, nval69, nval70, nval71, nval72, nval73, nval74, nval75, nval76, nval77, nval78, nval79, nval80, nval81, nval82, nval83, nval84, nval85, nval86, nval87, nval88, nval89, nval90, nval91, nval92, nval93, nval94, nval95, nval96, nval97, nval98, nval99, nval100, nval101, nval102, nval103, nval104, nval105, nval106, nval107, nval108, nval109, nval110, nval111, nval112, nval113, nval114, nval115, nval116, nval117, nval118, nval119, nval120, nval121, nval122, nval123, nval124, nval125, nval126, nval127, nval128, nval129, nval130, nval131, nval132, nval133, nval134, nval135, nval136, nval137, nval138, nval139, nval140, nval141, nval142, nval143, nval144, nval145, nval146, nval147, nval148, nval149, nval150, nval151, nval152, nval153, nval154, nval155, nval156, nval157, nval158, nval159, nval160, nval161, nval162, nval163, nval164, nval165, nval166, nval167, nval168, nval169, nval170, nval171, nval172, nval173, nval174, nval175, nval176, nval177, nval178, nval179, nval180, nval181, nval182, nval183, nval184, nval185, nval186, nval187, nval188, nval189, nval190, nval191, nval192, nval193, nval194, nval195, nval196, nval197, nval198, nval199, nval200, nval201, nval202, nval203, nval204, nval205, nval206, nval207, nval208, nval209, nval210, nval211, nval212, nval213, nval214, nval215, nval216, nval217, nval218, nval219, nval220, nval221, nval222, nval223, nval224, nval225, nval226, nval227, nval228, nval229, nval230, nval231, nval232, nval233, nval234, nval235, nval236, nval237, nval238, nval239, nval240, nval241, nval242, nval243, nval244, nval245, nval246, nval247, nval248, nval249, nval250, nval251, nval252, nval253, nval254, nval255, nval256, nval257, nval258, nval259, nval260, nval261, nval262, nval263, nval264, nval265, nval266, nval267, nval268, nval269, nval270, nval271, nval272, nval273, nval274, nval275, nval276, nval277, nval278, nval279, nval280, nval281, nval282, nval283, nval284, nval285, nval286, nval287, nval288, nval289, nval290, nval291, nval292, nval293, nval294, nval295, nval296, nval297, nval298, nval299, nval300, nval301, nval302, nval303, nval304, nval305, nval306, nval307, nval308, nval309, nval310, nval311, nval312, nval313, nval314, nval315, nval316, nval317, nval318, nval319, nval320, nval321, nval322, nval323, nval324, nval325, nval326, nval327, nval328, nval329, nval330, nval331, nval332, nval333, nval334, nval335, nval336, nval337, nval338, nval339, nval340, nval341, nval342, nval343, nval344, nval345, nval346, nval347, nval348, nval349, nval350, nval351, nval352, nval353, nval354, nval355, nval356, nval357, nval358, nval359, nval360, nval361, nval362, nval363, nval364, nval365, nval366, nval367, nval368, nval369, nval370, nval371, nval372, nval373, nval374, nval375, nval376, nval377, nval378, nval379, nval380, nval381, nval382, nval383, nval384, nval385, nval386, nval387, nval388, nval389, nval390, nval391, nval392, nval393, nval394, nval395, nval396, nval397, nval398, nval399, nval400, nval401, nval402, nval403, nval404, nval405, nval406, nval407, nval408, nval409, nval410, nval411, nval412, nval413, nval414, nval415, nval416, nval417, nval418, nval419, nval420, nval421, nval422, nval423, nval424, nval425, nval426, nval427, nval428, nval429, nval430, nval431, nval432, nval433, nval434, nval435, nval436, nval437, nval438, nval439, nval440, nval441, nval442, nval443, nval444, nval445, nval446, nval447, nval448, nval449, nval450, nval451, nval452, nval453, nval454, nval455, nval456, nval457, nval458, nval459, nval460, nval461, nval462, nval463, nval464, nval465, nval466, nval467, nval468, nval469, nval470, nval471, nval472, nval473, nval474, nval475, nval476, nval477, nval478, nval479, nval480, nval481, nval482, nval483, nval484, nval485, nval486, nval487, nval488, nval489, nval490, nval491, nval492, nval493, nval494, nval495, nval496, nval497, nval498, nval499, nval500, nval501, nval502, nval503, nval504, nval505, nval506, nval507, nval508, nval509, nval510, nval511, nval512, nval513, nval514, nval515, nval516, nval517, nval518, nval519, nval520, nval521, nval522, nval523, nval524, nval525, nval526, nval527, nval528, nval529, nval530, nval531, nval532, nval533, nval534, nval535, nval536, nval537, nval538, nval539, nval540, nval541, nval542, nval543, nval544, nval545, nval546, nval547, nval548, nval549, nval550, nval551, nval552, nval553, nval554, nval555, nval556, nval557, nval558, nval559, nval560, nval561, nval562, nval563, nval564, nval565, nval566, nval567, nval568, nval569, nval570, nval571, nval572, nval573, nval574, nval575, nval576, nval577, nval578, nval579, nval580, nval581, nval582, nval583, nval584, nval585, nval586, nval587, nval588, nval589, nval590, nval591, nval592, nval593, nval594, nval595, nval596, nval597, nval598, nval599, nval600, nval601, nval602, nval603, nval604, nval605, nval606, nval607, nval608, nval609, nval610, nval611, nval612, nval613, nval614, nval615, nval616, nval617, nval618, nval619, nval620, nval621, nval622, nval623, nval624, nval625, nval626, nval627, nval628, nval629, nval630, nval631, nval632, nval633, nval634, nval635, nval636, nval637, nval638, nval639, nval640, nval641, nval642, nval643, nval644, nval645, nval646, nval647, nval648, nval649, nval650, nval651, nval652, nval653, nval654, nval655, nval656, nval657, nval658, nval659, nval660, nval661, nval662, nval663, nval664, nval665, nval666, nval667, nval668, nval669, nval670, nval671, nval672, nval673, nval674, nval675, nval676, nval677, nval678, nval679, nval680, nval681, nval682, nval683, nval684, nval685, nval686, nval687, nval688, nval689, nval690, nval691, nval692, nval693, nval694, nval695, nval696, nval697, nval698, nval699, nval700, nval701, nval702, nval703, nval704, nval705, nval706, nval707, nval708, nval709, nval710, nval711, nval712, nval713, nval714, nval715, nval716, nval717, nval718, nval719, nval720, nval721, nval722, nval723, nval724, nval725, nval726, nval727, nval728, nval729, nval730, nval731, nval732, nval733, nval734, nval735, nval736, nval737, nval738, nval739, nval740, nval741, nval742, nval743, nval744, nval745, nval746, nval747, nval748, nval749, nval750, nval751, nval752, nval753, nval754, nval755, nval756, nval757, nval758, nval759, nval760, nval761, nval762, nval763, nval764, nval765, nval766, nval767, nval768, nval769, nval770, nval771, nval772, nval773, nval774, nval775, nval776, nval777, nval778, nval779, nval780, nval781, nval782, nval783, nval784, nval785, nval786, nval787, nval788, nval789, nval790, nval791, nval792, nval793, nval794, nval795, nval796, nval797, nval798, nval799, nval800, nval801, nval802, nval803, nval804, nval805, nval806, nval807, nval808, nval809, nval810, nval811, nval812, nval813, nval814, nval815, nval816, nval817, nval818, nval819, nval820, nval821, nval822, nval823, nval824, nval825, nval826, nval827, nval828, nval829, nval830, nval831, nval832, nval833, nval834, nval835, nval836, nval837, nval838, nval839, nval840, nval841, nval842, nval843, nval844, nval845, nval846, nval847, nval848, nval849, nval850, nval851, nval852, nval853, nval854, nval855, nval856, nval857, nval858, nval859, nval860, nval861, nval862, nval863, nval864, nval865, nval866, nval867, nval868, nval869, nval870, nval871, nval872, nval873, nval874, nval875, nval876, nval877, nval878, nval879, nval880, nval881, nval882, nval883, nval884, nval885, nval886, nval887, nval888, nval889, nval890, nval891, nval892, nval893, nval894, nval895, nval896, nval897, nval898, nval899, nval900, nval901, nval902, nval903, nval904, nval905, nval906, nval907, nval908, nval909, nval910, nval911, nval912, nval913, nval914, nval915, nval916, nval917, nval918, nval919, nval920, nval921, nval922, nval923, nval924, nval925, nval926, nval927, nval928, nval929, nval930, nval931, nval932, nval933, nval934, nval935, nval936, nval937, nval938, nval939, nval940, nval941, nval942, nval943, nval944, nval945, nval946, nval947, nval948, nval949, nval950, nval951, nval952, nval953, nval954, nval955, nval956, nval957, nval958, nval959, nval960, nval961, nval962, nval963, nval964, nval965, nval966, nval967, nval968, nval969, nval970, nval971, nval972, nval973, nval974, nval975, nval976, nval977, nval978, nval979, nval980, nval981, nval982, nval983, nval984, nval985, nval986, nval987, nval988, nval989, nval990, nval991, nval992, nval993, nval994, nval995, nval996, nval997, nval998, nval999, nval1000, nval1001, nval1002, nval1003, nval1004, nval1005, nval1006, nval1007, nval1008, nval1009, nval1010, nval1011, nval1012, nval1013, nval1014, nval1015, nval1016, nval1017, nval1018, nval1019, nval1020, nval1021, nval1022, nval1023, nval1024, nval1025, nval1026, nval1027, nval1028, nval1029, nval1030, nval1031, nval1032, nval1033, nval1034, nval1035, nval1036, nval1037, nval1038, nval1039, nval1040, nval1041, nval1042, nval1043, nval1044, nval1045, nval1046, nval1047, nval1048, nval1049, nval1050, nval1051, nval1052, nval1053, nval1054, nval1055, nval1056, nval1057, nval1058, nval1059, nval1060, nval1061, nval1062, nval1063, nval1064, nval1065, nval1066, nval1067, nval1068, nval1069, nval1070, nval1071, nval1072, nval1073, nval1074, nval1075, nval1076, nval1077, nval1078, nval1079, nval1080, nval1081, nval1082, nval1083, nval1084, nval1085, nval1086, nval1087, nval1088, nval1089, nval1090, nval1091, nval1092, nval1093, nval1094, nval1095, nval1096, nval1097, nval1098, nval1099, nval1100, nval1101, nval1102, nval1103, nval1104, nval1105, nval1106, nval1107, nval1108, nval1109, nval1110, nval1111, nval1112, nval1113, nval1114, nval1115, nval1116, nval1117, nval1118, nval1119, nval1120, nval1121, nval1122, nval1123, nval1124, nval1125, nval1126, nval1127, nval1128, nval1129, nval1130, nval1131, nval1132, nval1133, nval1134, nval1135, nval1136, nval1137, nval1138, nval1139, nval1140, nval1141, nval1142, nval1143, nval1144, nval1145, nval1146, nval1147, nval1148, nval1149, nval1150, nval1151, nval1152, nval1153, nval1154, nval1155, nval1156, nval1157, nval1158, nval1159, nval1160, nval1161, nval1162, nval1163, nval1164, nval1165, nval1166, nval1167, nval1168, nval1169, nval1170, nval1171, nval1172, nval1173, nval1174, nval1175, nval1176, nval1177, nval1178, nval1179, nval1180, nval1181, nval1182, nval1183, nval1184, nval1185, nval1186, nval1187, nval1188, nval1189, nval1190, nval1191, nval1192, nval1193, nval1194, nval1195, nval1196, nval1197, nval1198, nval1199, nval1200, nval1201, nval1202, nval1203, nval1204, nval1205, nval1206, nval1207, nval1208, nval1209, nval1210, nval1211, nval1212, nval1213, nval1214, nval1215, nval1216, nval1217, nval1218, nval1219, nval1220, nval1221, nval1222, nval1223, nval1224, nval1225, nval1226, nval1227, nval1228, nval1229, nval1230, nval1231, nval1232, nval1233, nval1234, nval1235, nval1236, nval1237, nval1238, nval1239, nval1240, nval1241, nval1242, nval1243, nval1244, nval1245, nval1246, nval1247, nval1248, nval1249, nval1250, nval1251, nval1252, nval1253, nval1254, nval1255, nval1256, nval1257, nval1258, nval1259, nval1260, nval1261, nval1262, nval1263, nval1264, nval1265, nval1266, nval1267, nval1268, nval1269, nval1270, nval1271, nval1272, nval1273, nval1274, nval1275, nval1276, nval1277, nval1278, nval1279, nval1280, nval1281, nval1282, nval1283, nval1284, nval1285, nval1286, nval1287, nval1288, nval1289, nval1290, nval1291, nval1292, nval1293, nval1294, nval1295, nval1296, nval1297, nval1298, nval1299, nval1300, nval1301, nval1302, nval1303, nval1304, nval1305, nval1306, nval1307, nval1308, nval1309, nval1310, nval1311, nval1312, nval1313, nval1314, nval1315, nval1316, nval1317, nval1318, nval1319, nval1320, nval1321, nval1322, nval1323, nval1324, nval1325, nval1326, nval1327, nval1328, nval1329, nval1330, nval1331, nval1332, nval1333, nval1334, nval1335, nval1336, nval1337, nval1338, nval1339, nval1340, nval1341, nval1342, nval1343, nval1344, nval1345, nval1346, nval1347, nval1348, nval1349, nval1350, nval1351, nval1352, nval1353, nval1354, nval1355, nval1356, nval1357, nval1358, nval1359, nval1360, nval1361, nval1362, nval1363, nval1364, nval1365, nval1366, nval1367, nval1368, nval1369, nval1370, nval1371, nval1372, nval1373, nval1374, nval1375, nval1376, nval1377, nval1378, nval1379, nval1380, nval1381, nval1382, nval1383, nval1384, nval1385, nval1386, nval1387, nval1388, nval1389, nval1390, nval1391, nval1392, nval1393, nval1394, nval1395, nval1396, nval1397, nval1398, nval1399, nval1400, nval1401, nval1402, nval1403, nval1404, nval1405, nval1406, nval1407, nval1408, nval1409, nval1410, nval1411, nval1412, nval1413, nval1414, nval1415, nval1416, nval1417, nval1418, nval1419, nval1420, nval1421, nval1422, nval1423, nval1424, nval1425, nval1426, nval1427, nval1428, nval1429, nval1430, nval1431, nval1432, nval1433, nval1434, nval1435, nval1436, nval1437, nval1438, nval1439, nval1440, nval1441, nval1442, nval1443, nval1444, nval1445, nval1446, nval1447, nval1448, nval1449, nval1450, nval1451, nval1452, nval1453, nval1454, nval1455, nval1456, nval1457, nval1458, nval1459, nval1460, nval1461, nval1462, nval1463, nval1464, nval1465, nval1466, nval1467, nval1468, nval1469, nval1470, nval1471, nval1472, nval1473, nval1474, nval1475, nval1476, nval1477, nval1478, nval1479, nval1480, nval1481, nval1482, nval1483, nval1484, nval1485, nval1486, nval1487, nval1488, nval1489, nval1490, nval1491, nval1492, nval1493, nval1494, nval1495, nval1496, nval1497, nval1498, nval1499, nval1500, nval1501, nval1502, nval1503, nval1504, nval1505, nval1506, nval1507, nval1508, nval1509, nval1510, nval1511, nval1512, nval1513, nval1514, nval1515, nval1516, nval1517, nval1518, nval1519, nval1520, nval1521, nval1522, nval1523, nval1524, nval1525, nval1526, nval1527, nval1528, nval1529, nval1530, nval1531, nval1532, nval1533, nval1534, nval1535, nval1536, nval1537, nval1538, nval1539, nval1540, nval1541, nval1542, nval1543, nval1544, nval1545, nval1546, nval1547, nval1548, nval1549, nval1550, nval1551, nval1552, nval1553, nval1554, nval1555, nval1556, nval1557, nval1558, nval1559, nval1560, nval1561, nval1562, nval1563, nval1564, nval1565, nval1566, nval1567, nval1568, nval1569, nval1570, nval1571, nval1572, nval1573, nval1574, nval1575, nval1576, nval1577, nval1578, nval1579, nval1580, nval1581, nval1582, nval1583, nval1584, nval1585, nval1586, nval1587, nval1588, nval1589, nval1590, nval1591, nval1592, nval1593, nval1594, nval1595, nval1596, nval1597, nval1598, nval1599, nval1600, nval1601, nval1602, nval1603, nval1604, nval1605, nval1606, nval1607, nval1608, nval1609, nval1610, nval1611, nval1612, nval1613, nval1614, nval1615, nval1616, nval1617, nval1618, nval1619, nval1620, nval1621, nval1622, nval1623, nval1624, nval1625, nval1626, nval1627, nval1628, nval1629, nval1630, nval1631, nval1632, nval1633, nval1634, nval1635, nval1636, nval1637, nval1638, nval1639, nval1640, nval1641, nval1642, nval1643, nval1644, nval1645, nval1646, nval1647, nval1648, nval1649, nval1650, nval1651, nval1652, nval1653, nval1654, nval1655, nval1656, nval1657, nval1658, nval1659, nval1660, nval1661, nval1662, nval1663, nval1664, nval1665, nval1666, nval1667, nval1668, nval1669, nval1670, nval1671, nval1672, nval1673, nval1674, nval1675, nval1676, nval1677, nval1678, nval1679, nval1680, nval1681, nval1682, nval1683, nval1684, nval1685, nval1686, nval1687, nval1688, nval1689, nval1690, nval1691, nval1692, nval1693, nval1694, nval1695, nval1696, nval1697, nval1698, nval1699, nval1700, nval1701, nval1702, nval1703, nval1704, nval1705, nval1706, nval1707, nval1708, nval1709, nval1710, nval1711, nval1712, nval1713, nval1714, nval1715, nval1716, nval1717, nval1718, nval1719, nval1720, nval1721, nval1722, nval1723, nval1724, nval1725, nval1726, nval1727, nval1728, nval1729, nval1730, nval1731, nval1732, nval1733, nval1734, nval1735, nval1736, nval1737, nval1738, nval1739, nval1740, nval1741, nval1742, nval1743, nval1744, nval1745, nval1746, nval1747, nval1748, nval1749, nval1750, nval1
```

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

- $\Psi(r_1, \dots, r_N)$
- $\Delta_i \Psi(r_1, \dots, r_i, \dots, r_N)$: kinetic energy
- $\nabla_i \Psi(r_1, \dots, r_i, \dots, r_N)$: drift in the stochastic process

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

- $\Psi(r_1, \dots, r_N)$
- $\Delta_i \Psi(r_1, \dots, r_i, \dots, r_N)$: kinetic energy
- $\nabla_i \Psi(r_1, \dots, r_i, \dots, r_N)$: drift in the stochastic process

Kernels implemented and well tested today

- AOs: $\chi(r)$, $\nabla \chi(r)$, $\Delta \chi(r)$
- MOs: $\varphi(r)$, $\nabla \varphi(r)$, $\Delta \varphi(r)$
- Jastrow correlation factor (eN, ee, eeN)
- Inverses of small matrices

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

- $\Psi(r_1, \dots, r_N)$
- $\Delta_j \Psi(r_1, \dots, r_i, \dots, r_N)$: kinetic energy
- $\nabla_j \Psi(r_1, \dots, r_i, \dots, r_N)$: drift in the stochastic process

Kernels implemented and well tested today

- AOs: $\chi(r)$, $\nabla \chi(r)$, $\Delta \chi(r)$
- MOs: $\varphi(r)$, $\nabla \varphi(r)$, $\Delta \varphi(r)$
- Jastrow correlation factor (eN, ee, eeN)
- Inverses of small matrices

Work in progress

- Everything else required to compute Ψ , $\nabla \Psi$ and $\Delta \Psi$.

Algorithms

Before computing anything, QMCKI needs to be given a trial wave function.

Setting wave function parameters

- Wave function exchange between codes is a major difficulty
- Our solution:
 - Define a standard format for wavefunction parameters
 - **TREXIO**: TREX Input/Output library (see Evgeny Posenitskiy's presentation)

Initialization of QMCKI

Two ways:

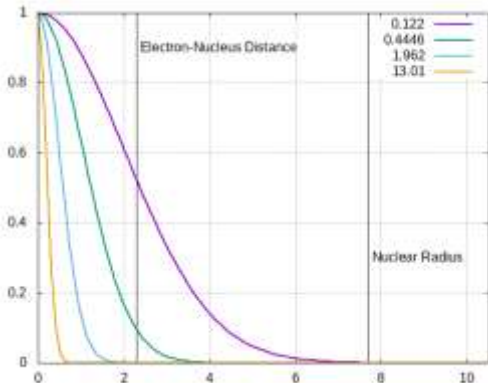
- 1 **Control** : Each array can be set by hand
- 2 **Simplicity** : Read all the wave function parameters from a TREXIO file

Atomic Orbitals

$$R_S(r) = N_S |r - R_A|^{n_S} \sum_{k=1}^{N_{\text{prim}}} a_{kS} f_{kS} \exp(-\gamma_{kS} |r - R_A|^p).$$

Flexible

- Software like GAMESS use different normalization factors for d orbitals
- Implementing Slater-type orbitals is a minor modification (in the very long to-do list)
- Contribution from the FHI-AIMS group for the evaluation of numerical AOs
- Separation of the radial and angular components packed in shells
- Efficient computation of powers of x, y, z to maximize data re-use



- Definition of an atomic radius for each nucleus beyond which all AOs are zero (VGL^a).
- Primitives are sorted in ascending order of the exponents.
- Only non-zero elements are computed

^aVGL: value, gradients, Laplacian

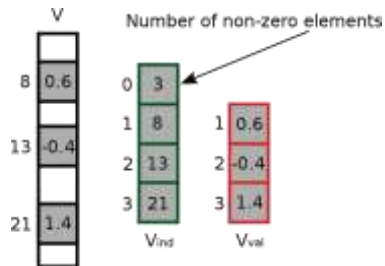
Molecular Orbitals

$$\begin{aligned} \varphi_i(r_j) &= \sum_k A_{ik} \chi_k(r_j) & B_1 &= A \cdot C_1 \\ \nabla_x \varphi_i(r_j) &= \sum_k A_{ik} \nabla_x \chi_k(r_j) & B_2 &= A \cdot C_2 \\ \nabla_y \varphi_i(r_j) &= \sum_k A_{ik} \nabla_y \chi_k(r_j) & B_3 &= A \cdot C_3 \\ \nabla_z \varphi_i(r_j) &= \sum_k A_{ik} \nabla_z \chi_k(r_j) & B_4 &= A \cdot C_4 \\ \Delta \varphi_i(r_j) &= \sum_k A_{ik} \Delta \chi_k(r_j) & B_5 &= A \cdot C_5 \end{aligned}$$

- QMC=Chem (2013): <https://doi.org/10.1002/jcc.23216>
- Exploits the common sparse character of the AO matrices:
 - When $\chi(r) = 0$ because r is too far, all the derivatives are also zero
 - Quadratic scaling
- Can be fully vectorized
 - >60% of peak performance on Sandy-Bridge CPUs

	Smallest system	β -Strand	β -Strand TZ	1ZE7	1AMB
N	158	434	434	1056	1731
N_{basis}	404	963	2934	2370	3892
% of non-zero ^a MO coefficients a_{ij} ($A_{ij} \neq 0$)	81.3% (99.4%)	48.4% (76.0%)	73.4% (81.9%)	49.4% (72.0%)	37.1% (66.1%)
Average % of non-zero basis functions $\chi_i(\mathbf{r}_j)$ ($B_{1ij} \neq 0$)	36.2%	14.8%	8.2%	5.7%	3.9%
Average number of non-zero elements per column of B_{1ij}	146	142	241	135	152

```
1  do j=1,point_num
2      mo_vgl(:, :, j) = 0. d0
3      do k=1, ao_num
4          if (ao_vgl(k, 1, j) /= 0. d0) then
5              c1 = ao_vgl(k, 1, j)
6              c2 = ao_vgl(k, 2, j)
7              c3 = ao_vgl(k, 3, j)
8              c4 = ao_vgl(k, 4, j)
9              c5 = ao_vgl(k, 5, j)
10             do i=1, mo_num
11                 mo_vgl(i, 1, j) = mo_vgl(i, 1, j) + coefficient_t(i, k) * c1
12                 mo_vgl(i, 2, j) = mo_vgl(i, 2, j) + coefficient_t(i, k) * c2
13                 mo_vgl(i, 3, j) = mo_vgl(i, 3, j) + coefficient_t(i, k) * c3
14                 mo_vgl(i, 4, j) = mo_vgl(i, 4, j) + coefficient_t(i, k) * c4
15                 mo_vgl(i, 5, j) = mo_vgl(i, 5, j) + coefficient_t(i, k) * c5
16             end do
17         end if
18     end do
19 end do
```

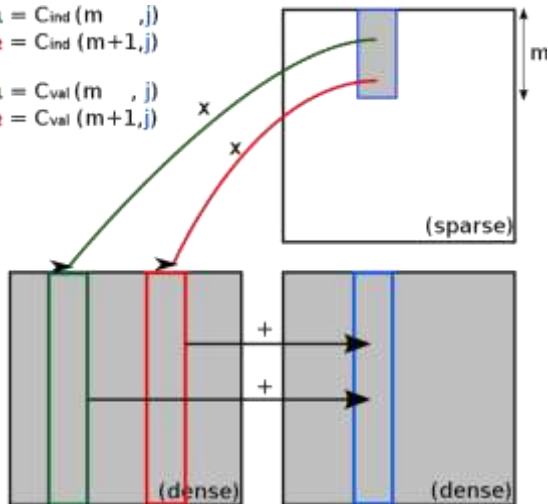


$k_1 = C_{ind}(m, j)$

$k_2 = C_{ind}(m+1, j)$

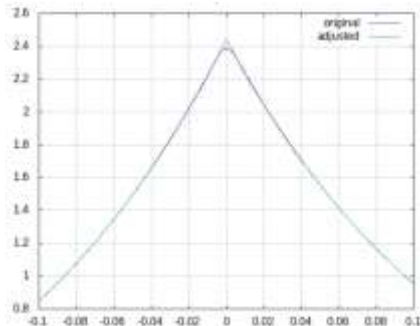
$c_1 = C_{val}(m, j)$

$c_2 = C_{val}(m+1, j)$



$$\varphi_{\text{cusp}i}(r) = \varphi_i(r) - \varphi_{\text{SA}i}(r) + \sum_{k=0}^3 f_k |r - R_A|^k, \quad \text{where } |r - R_A| < r_{\text{cusp},A}$$

- $\varphi_{\text{SA}i}$: contributions of the s AOs centered at A to MO φ_i .
- 3 conditions:
 - Electron-nucleus cusp at $|r - R_A| = 0$
 - Continuity of the MO: $\varphi_{\text{cusp}i} = \varphi_i$ when $|r - R_A| = r_{\text{cusp},A}$
 - Continuity of the gradient: $\nabla \varphi_{\text{cusp}i}(r) = \nabla \varphi_i(r)$ when $|r - R_A| = r_{\text{cusp},A}$



Jastrow factor

$$J_{\text{Jen}}(\mathbf{r}, \mathbf{R}) = \prod_{a=1}^{N_{\text{nucl}}} \prod_{i=1}^{N_{\text{elec}}} \sum_{j=1}^{N_{\text{elec}}} \sum_{p=2}^{N_{\text{nucl}}} \sum_{k=0}^{p-1} \sum_{l=0}^{p-k} 2\delta_{k,0} C_{lkpa} (r_{ij})^k (R_{ia})^l + (R_{ja})^l (R_{ia} R_{ja})^{(p-k-l)/2}$$

can be rewritten as

$$J_{\text{Jen}}(\mathbf{r}, \mathbf{R}) = \prod_{p=2}^{N_{\text{nucl}}} \prod_{k=0}^{p-1} \sum_{l=0}^{p-k} 2\delta_{k,0} \prod_{a=1}^{N_{\text{nucl}}} \prod_{i=1}^{N_{\text{elec}}} C_{lkpa} \bar{R}_{i,a,(p-k-l)/2} \bar{P}_{i,k,a,(p-k+l)/2} \quad (\downarrow \text{complexity})$$

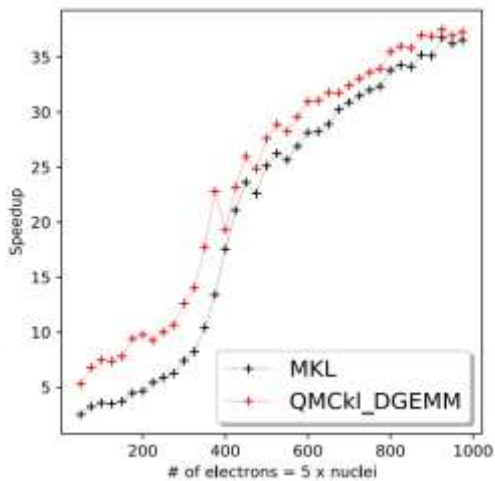
with

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

$$\begin{aligned}
 \nabla_{im} J_{\text{Jeen}}(r, R) = & \sum_{\text{bond}} \sum_{l=1}^{p-k} \sum_{k=0}^{2\delta_{k,0}} \sum_{\text{nucl}} C_{lkpa} \sum_{\text{elec}} \bar{G}_{i,m,a,(p-k-l)/2} \bar{P}_{i,a,k,(p-k+l)/2} + \\
 & \bar{G}_{i,m,a,(p-k+l)/2} \bar{P}_{i,a,k,(p-k-l)/2} + \bar{R}_{i,a,(p-k-l)/2} \bar{Q}_{i,m,a,k,(p-k+l)/2} + \\
 & \bar{R}_{i,a,(p-k+l)/2} \bar{Q}_{i,m,a,k,(p-k-l)/2} + \delta_{m,4} \\
 & \bar{G}_{i,1,a,(p-k+l)/2} \bar{Q}_{i,1,a,k,(p-k-l)/2} + \bar{G}_{i,2,a,(p-k+l)/2} \bar{Q}_{i,2,a,k,(p-k-l)/2} + \\
 & \bar{G}_{i,3,a,(p-k+l)/2} \bar{Q}_{i,3,a,k,(p-k-l)/2} + \bar{G}_{i,1,a,(p-k-l)/2} \bar{Q}_{i,1,a,k,(p-k+l)/2} + \\
 & \bar{G}_{i,2,a,(p-k-l)/2} \bar{Q}_{i,2,a,k,(p-k+l)/2} + \bar{G}_{i,3,a,(p-k-l)/2} \bar{Q}_{i,3,a,k,(p-k+l)/2}
 \end{aligned}$$

with

$$\bar{G}_{i,m,a,l} = \frac{\partial (R_{ia})}{\partial r_i}, \quad \bar{g}_{i,m,j,k} = \frac{\partial (r_{ij})}{\partial r_i}, \quad \text{and } \bar{Q}_{i,m,a,k,l} = \sum_{j=1}^{\text{elec}} \bar{g}_{i,m,j,k} \bar{R}_{j,a,l}$$



HPC implementations

- MAQAO, developed by the UVSQ team, is used to help us optimize the CPU code
 - Loop-level diagnostics
 - Vectorization ratio
 - Hints to improve efficiency
- Algorithms rewritten in C:
 - C compilers are usually more mature than Fortran on new hardware
 - Access to more low-level features than Fortran (pinned memory, alignment, inline assembly, etc)
- Precision can be changed on-the-fly: switch to single-precision if possible
- Specialization:
 - Specialization for s , p and d AOs
 - Inverse of small matrices hard-coded for 2×2 to 5×5
 - Small matrix multiplication
 - ...

```

subroutine cofactor(n,lda,r,ldb,ldu,ldv)
  implicit none
  integer n,lda,r,ldb,ldu,ldv
  double precision, intent(in) :: A(lda,n)
  double precision, intent(out) :: B(lda,n)
  integer*4, intent(in) :: lda, ldb
  integer*4, intent(out) :: nr
  double precision, intent(out) :: det_1
  integer :: i,j
  det_1 = a(1,1)+a(1,2)+a(1,3)+a(1,4)-a(2,4)-a(4,3)
  B(1,1) = a(1,2)+a(1,3)+a(1,4)-a(2,4)-a(4,3)
  B(1,2) = a(1,1)+a(1,3)+a(1,4)-a(2,4)-a(4,3)
  B(1,3) = a(1,1)+a(1,2)+a(1,4)-a(2,4)-a(4,3)
  B(1,4) = a(1,1)+a(1,2)+a(1,3)-a(2,4)-a(4,3)
  B(2,1) = a(2,1)+a(2,3)+a(2,4)-a(1,4)-a(4,3)
  B(2,2) = a(2,1)+a(2,3)+a(2,4)-a(1,4)-a(4,3)
  B(2,3) = a(2,1)+a(2,3)+a(2,4)-a(1,4)-a(4,3)
  B(2,4) = a(2,1)+a(2,3)+a(2,4)-a(1,4)-a(4,3)
  B(3,1) = a(3,1)+a(3,2)+a(3,4)-a(2,4)-a(4,3)
  B(3,2) = a(3,1)+a(3,2)+a(3,4)-a(2,4)-a(4,3)
  B(3,3) = a(3,1)+a(3,2)+a(3,4)-a(2,4)-a(4,3)
  B(3,4) = a(3,1)+a(3,2)+a(3,4)-a(2,4)-a(4,3)
  B(4,1) = a(4,1)+a(4,2)+a(4,3)-a(2,4)-a(4,3)
  B(4,2) = a(4,1)+a(4,2)+a(4,3)-a(2,4)-a(4,3)
  B(4,3) = a(4,1)+a(4,2)+a(4,3)-a(2,4)-a(4,3)
  B(4,4) = a(4,1)+a(4,2)+a(4,3)-a(2,4)-a(4,3)
  nr = n
end subroutine cofactor
    
```

- GPU library has the same functions, suffixed with `_device`
- Two different flavours: OpenMP or OpenACC
- Possibility to use CPU and GPU library together in the same code
- In early development, not fully integrated to our codes yet (work in progress)
- Although the kernels are fast on Nvidia GPUs, GPU acceleration is not clear because of data transfer
 - Maybe efficient on next generation of hardware
- On GPU, brute-force CuBLAS DGEMM is faster than sparse AO-MO transformation. Energy efficiency?

- Tensor core instructions are not generated in OpenMP kernels $\Rightarrow \leq 50\%$ peak DP
- Conflict between OpenMP runtime of the code and of QMCKI-GPU \Rightarrow
 - Need to compile the code with GPU compiler (Nvfortran)
 - May not compile, or with low CPU efficiency
 - Our solution: decouple QMCKI-CPU and QMCKI-GPU and recover CPU performance with QMCKI-CPU
- RocBLAS \sim CuBLAS, but some OpenMP kernels have $10\times$ lower performance on AMD GPUs than Nvidia (under investigation. . .)
- Unreliable software stack: \Rightarrow Compared to CPU, very inefficient in human resources
- Open Question:
 - Should we have opted instead for vendor-specific implementations? (Cuda, HIP)

```
1 $ tar -zxvf qmckl.tar.gz
2 $ cd qmckl
3 $ ./configure --enable-hpc
4 $ make -j 32
5 $ make check
6 $ make install
```

- QMCKl has been used in

- C/ C++
- Fortran
- Python
- Julia
- Rust

- Very few dependencies:
 - BLAS/Lapack (CPU)
 - TRESIO (optional) with HDF5 (optional)
- BSD license: very permissive. You can distribute the `tar.gz` with your code
- Hosted on GitHub:
<https://github.com/trex-coe/qmckl>

Integration into TREN codes

- Single-core benchmark: C₆₀, Hartree-Fock/cc-pVQZ/ECP(BFD)
 - Time for a single MC step (all-electrons)
 - 4140 AOs, 120 MOs, 240 electrons

CPU	Compiler	QMCKl	milliseconds	Speedup
Intel(R) Core(TM) i7 (8-core Laptop, 2.8GHz)	ifort/mkl	-	24.58	
	ifort/mkl	gcc12	24.06	1.02x
	ifort/mkl	icx	23.85	1.03x
	gfortran/openblas	-	30.58	
	gfortran/openblas	gcc12	26.04	1.17x
ARM Neoverse V1 (80 cores, 3GHz)	gfortran/armpl	-	41.24	
	gfortran/armpl	gcc12	31.91	1.29x

- Single-core benchmark: C₆₀, Hartree-Fock/cc-pVXZ/ECP(BFD)
 - Short VMC run
 - 4140 AOs, 120 MOs, 240 electrons

Basis	# AOs	Compiler	QMCKl	seconds	Speedup
cc-pVDZ	840	ifort/mkl	-	315.45	
			gcc12	218.29	1.45x
			icx	212.35	1.49x
cc-pVTZ	2040	ifort/mkl	-	565.67	
			gcc12	287.32	1.97x
			icx	271.68	2.08x
cc-pVQZ	4140	ifort/mkl	-	993.42	
			gcc12	462.74	2.15x
			icx	441.32	2.25x

- Reproducibility of QMC calculations (Jastrow factors)
- 3D visualization software:
 - AO or MO visualization
 - Interpretative methods like AIM or ELF
- Numerical integration
 - Computation of density grids for DFT with gradients
 - Jastrow factor in transcorrelated methods (Quantum Package)
- Teaching QMC algorithms in Jupyter notebooks
- Implementation of QMC methods in traditional quantum chemistry software

```

1  import qmckl
2  import numpy as np
3
4  def main(trexio_filename):
5      context = qmckl.context_create()           # Create a QMCKl context
6      qmckl.trexio_read(context, trexio_filename) # Read the TREXIO file into the context
7
8      nucl_num  = qmckl.get_nucleus_num(context) # Get the number of nuclei
9      nucl_coord = qmckl.get_nucleus_coord(context, 'N', nucl_num*3) # Get the nuclear coordinates
10     nucl_coord = np.reshape(nucl_coord, (3, nucl_num))
11     mo_num     = qmckl.get_mo_basis_mo_num(context) # Get the number of MOs
12
13     point      = setup_grid_points(nucl_coord)
14     point_num  = len(point)
15
16     qmckl.set_point(context, 'N', point_num, np.reshape(point, (point_num*3))) # Give points to QMCKl
17
18     mo_value = qmckl.get_mo_basis_mo_value(context, point_num*mo_num) # Get the values of the MOs
19
20     qmckl.context_destroy(context)           # Free QMCKl resources

```

CNRS

- Vijay Gopal Chilkuri
- Evgeny Posenitskiy
- Anthony Scemama

U-Twente

- Ravindra Shinde
- Edgar Landinez Borda
- Ramon Lorenzo Panades-Barrueta
- Claudia Filippi

SISSA

- Oto Kohulak
- Sandro Sorella

CINECA

- Tommaso Gorni
- Gianfranco Abrusci

UVSQ

- François Coppens
- Kevin Camus
- Aurelien Delval
- Max Hoffer
- Pablo Heitor De Oliveira Castro Herrero
- Cedric Valensi
- William Jalby