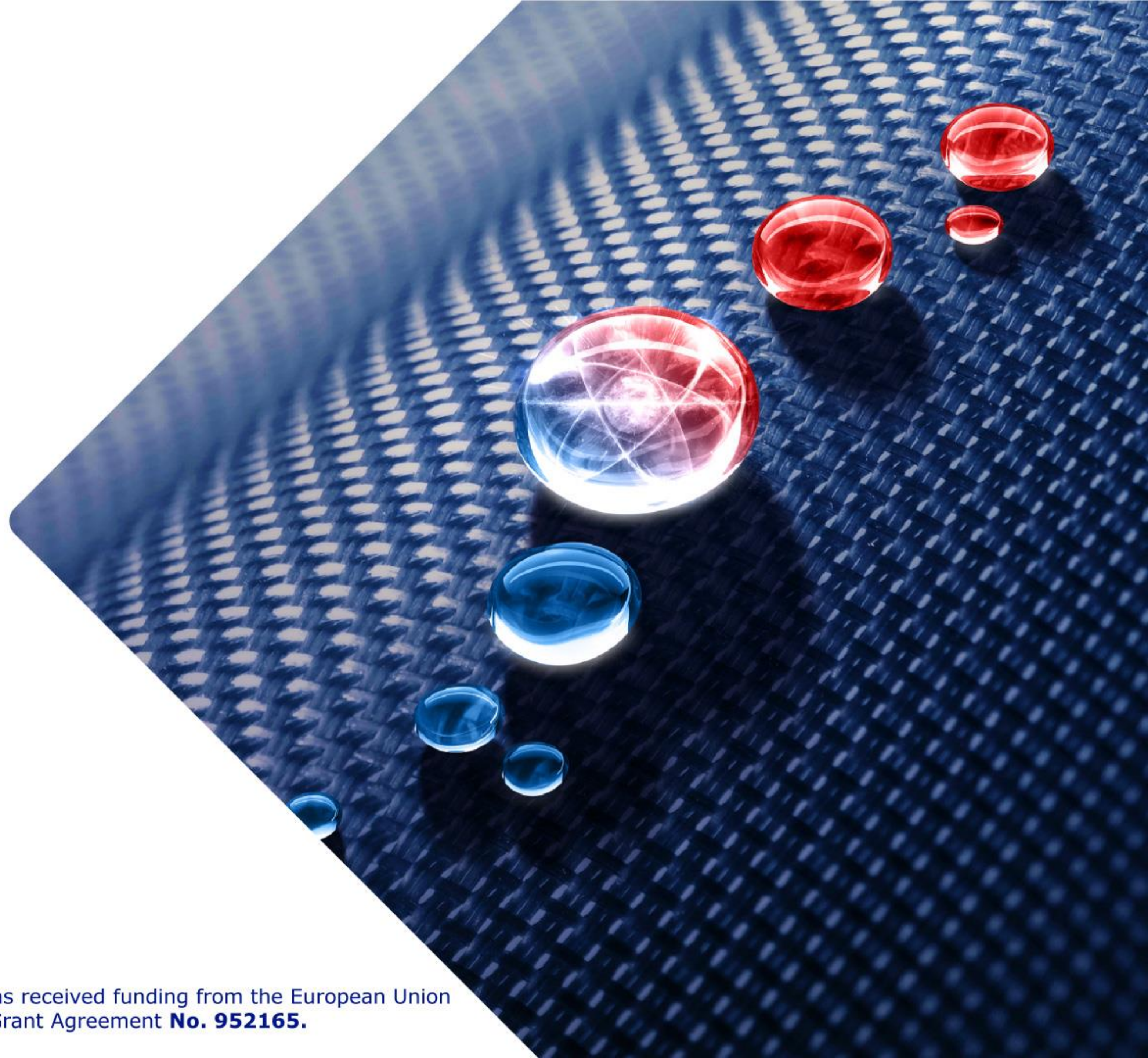
Targeting Real chemical accuracy at the EXascale

CO DESIGNING A HIGH PERFORMANCE AND PORTABLE LIBRARY (QMCKL): ONE OF THE MAJOR CHALLENGES ADDRESSED BY TREX CoE

A. Scemama, V.G. Chilkuri (Univ Toulouse),
P. De Oliveira, C. Valensi, W. Jalby (UVSQ)

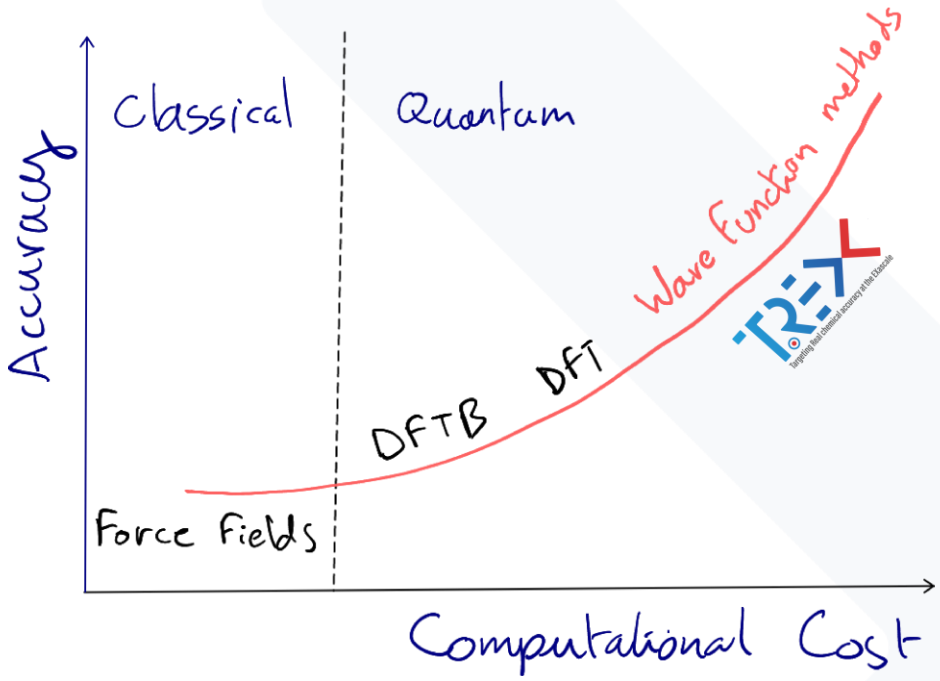


Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizoon 2020 research and innovation programme under Grant Agreement **No. 952165.**

TREX Mission: To develop, promote, and maintain open-source, exascale-ready software solutions in (stochastic) quantum chemistry

Materials modeling at the nanoscale with extreme accuracy

- Scientists in quantum chemistry, physics, and machine learning
- Software and HPC experts
- Tech and communication SMEs
- Representative of user communities



What: Use Quantum Monte Carlo Methods

- Highly accurate
- Massively parallelizable (multiple QMC trajectories)
- No Blocking communications
- CPU intensive (difficult to exploit)

Objective: Make codes ready for exascale

How: Instead of re-writing codes, provide libraries

- A library for exchanging information between codes:

TREXIO ➡ Enables HTC

- A library for high-performance numerical computation:

QMCkl ➡ Enables HPC

Create a platform of interoperable flagship codes extendable and/or operable with codes outside TREX

Objectives

1. **PRODUCTIVITY**: Used and developed by scientists, can be called from different languages
2. **PORTABILITY:** available on large number of hardware and software platforms
3. **PERFORMANCE:** Must be efficient

"Classical" challenge : be good simultaneously on the 3 objectives. There is a workshop at SC devoted to this triple objective.

We need to go beyond these classical objectives:

- Not only performance but also energy consumption should be a major goal
- Numerical accuracy is extremely important

Major axes/guidelines

1. **Focus and Specialization**: focused on well identified objectives (not general)
2. **Ease of interaction with software and hardware environment:** beyond classic portability
3. **Use of Advanced Tuning Tools:** adaptable library via tools instead of static library

Objectives

1. **Application target = QMC**: use of high degree of parallelism present in most QMC applications. Focus on single node/core implementations
2. **Focus on a few reference platforms:** CPU (ISA: X86 + ARM Neoverse), GPU
3. **Specific needs of our target applications:** for example no need of general arbitrary size Matrix Multiplies. Many of our apps are using Rank K Updates: (MxK) x (KxM) with K much smaller than M

GOOD NEWS:

- For our propose (scientific computing) X86 and ARM general architecture share a lot of common characteristics with secondary differences which can be dealt with automatically
- CPU and GPU all strongly require vectors…. But memory constraints are very different

Objectives

1. **Interaction with context**: routines available in source form so compiler can inline and optimize through calls
2. **Develop optimized but generic code versions:** use tools for generating highly optimized and specific version
3. **Strongly structure arrays within the library:** systematic use of tiled arrays for improving memory hierarchy usage
4. **Systematic use of (1+n) library versions:** a pedagogical/reference version and several optimized versions
5. **OPEN SOURCE + STANDARDS:** easy to modify and integrate. Strong use of standard: OpenMP directives for GPU and vectorization

Objectives

1. **Obtain high performance across a large range of platforms (first CPU)**
2. **Provide input for compilers/library/hardware designers**

Approach

- Start from the generic version perform first level (generic) optimization following tools guidance
- Perform detailed analysis of hardware and software interaction (including low level)
- Use tools (MAQAO) to automate info gathering and performance comparison
- Use tools (MAQAO) for generating specialized versions in partocular auto tuners for last mile optimization

Key issues analyzed

1. **Profile categorizations:** time spent in libraries, binary, loops (innermost/outermost), etc…
2. **Flow complexity:** number of paths, presence of calls, etc…
3. **Array access:** unit/non unit stride access, indirect
4. **Vectorization:** not only amount of vector instruictions but also assess vectorization quality

IMPORTANT:

- All of the above analysis is performed at the ASM level (either statically or dynamically at run time)
- This analysis depends upon compiler and processor used

GOOD NEWS: we can test various compilers and hardware and perform comparative studies. Very usefull for vectorization.

Provide performance estimates when specific optimizations are triggered

1. **Perfect OpenMP/MPI/Pthread:** suppress time spent in these parallelism libraries
2. **Perfect OpenMP/MPI/Pthread + Perfect Load balancing:** suppress time spent in these parallelism libraries + perform perfect load balancing
3. **Perfect compiler**: gets rid of all of the “integer” operations
4. **Perfect arithmetic FP vectorization:** assumes arithmetic FP vectorization
5. **Perfect Full vectorization**: assumes arithmetic FP + Load/Store vectorization
6. **L1 data access:** assumes that all data access are performed from L1

IMPORTANT: all of these performance estimates are computed at the loop level but their performance impact is extrapolated at the whole application.

Target: Unicore Skylake Xeon(R) Platinum 8170 + ICC/IFORT 2021 –O3

▶ Colums Filter

Loop id	Source Location	Source Function	Level	Coverage run_0 (%)	Vectorization Ratio (%)	Vectorization Efficiency (%)	Speedup If No Scalar Integer	Speedup If FP Vectorized	Speedup If Fully Vectorized	Speedup If Perfect Load Balancing run_0	Stride 0	Stride 1	Stride n	Stride Unknown
703	libqmckl.so.0 - qmckl_jastrow_f.F90:2088-2105 [...]	qmckl_compute_factor_een_deriv_e_f	Innermost	12.56	100	50	1.3	1.21	2	1	1	12	0	1
602	libqmckl.so.0 - qmckl_jastrow_f.F90:798-815	qmckl_compute_factor_een_rescaled_e_deriv_e_f	Innermost	7.74	55.56	30.56	1	1	3.55	1	0	6	0	0
640	libqmckl.so.0 - qmckl_jastrow_f.F90:1050-1067	qmckl_compute_factor_een_rescaled_n_deriv_e_f	Innermost	2.15	56.76	31.08	1	1	3.42	1	0	6	0	0
155	libqmckl.so.0 - qmckl_jastrow.c:1649-1653	qmckl_compute_een_rescaled_e_hpc	Innermost	1.98	0	12.5	1	1	8	1	0	2	0	2
603	libqmckl.so.0 - qmckl_jastrow_f.F90:798-815	qmckl_compute_factor_een_rescaled_e_deriv_e_f	Innermost	1.29	0	12.5	1	1	8	1	0	2	0	4
606	libqmckl.so.0 - qmckl_jastrow_f.F90:784-789	qmckl_compute_factor_een_rescaled_e_deriv_e_f	Innermost	1.17	100	44.74	1.02	1.35	2.46	1	0	5	0	0
161	libqmckl.so.0 - qmckl_jastrow.c:1630-1633	qmckl_compute_een_rescaled_e_hpc	Innermost	0.39	0	12.5	1.2	1.5	8	1	0	0	0	0

MAQAO ANLYSIS OF JASTROW ROUTINE: GLOBAL LEVEL

Target: Unicore Skylake Xeon(R) Platinum 8170 + ICC/IFORT 2021 –O3

Global Metrics

Total Time (s)		54.83
Profiled Time (s)		52.76
Time in analyzed loops (%)		30.9
Time in analyzed innermost loops (%)		29.8
Time in user code (%)		31.3
Compilation Options		OK
Perfect Flow Complexity		1.00
Iterations Count		1.00
Array Access Efficiency (%)		92.8
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.04
	Nb Loops to get 80%	2
FP Vectorised	Potential Speedup	1.03
	Nb Loops to get 80%	3
Fully Vectorised	Potential Speedup	1.25
	Nb Loops to get 80%	6
Data In L1 Cache	Potential Speedup	1.10
	Nb Loops to get 80%	1
FP Arithmetic Only	Potential Speedup	1.16
	Nb Loops to get 80%	7

Target: Unicore Skylake Xeon(R) Platinum 8170 + ICC/IFORT 2021 –O3

- [x] Analysis
- [ ] CQA speedup if no scalar integer
- [ ] CQA speedup if FP arith vectorized
- [ ] CQA speedup if fully vectorized
- [x] Number of paths
- [ ] Vectorization Ratio (%)
- [ ] Vectorization Efficiency (%)
- [x] ORIG / DL1
- [ ] Saturation ratio (MAX(DL1,LS)/REF)
- [x] Saturation
- [x] FP/CQA(FP)
- [x] DL1/CQA(DL1)
- [x] FP/LS
- [x] ORIG (cycles per iteration)
- [x] STA (ORIG)
- [x] REF (cycles per iteration)
- [x] STA (REF)
- [x] FP (cycles per iteration)
- [x] STA (FP)
- [x] LS (cycles per iteration)
- [x] STA (LS)
- [x] DL1 (cycles per iteration)
- [x] STA (DL1)
- [x] FES (cycles per iteration)
- [x] STA (FES)
- [x] CQA cycles
- [x] CQA cycles if no scalar integer
- [x] CQA cycles if FP arith vectorized
- [x] CQA cycles if fully vectorized
- [x] Iteration count
- [x] Function
- [x] Source
- [x] Nb FP_ADD / CPI
- [x] Nb FP_MUL / CPI
- [x] CAP(FP)
- [x] BW(FP)
- [x] SAT(FP)
- [x] CAP(L1R)
- [x] BW(L1R)
- [x] SAT(L1R)
- [x] CAP(L1W)
- [x] BW(L1W)
- [x] SAT(L1W)
- [x] CAP(L2)
- [x] BW(L2)
- [x] SAT(L2)
- [x] CAP(L3)
- [x] BW(L3)
- [x] SAT(L3)
- [x] CAP(RAM_R)
- [x] CAP(RAM_W)

Select none

ID	Module	Coverage (% app. time)	Analysis	Number of paths	ORIG / DL1	Saturation	FP/CQA(FP)	DL1/CQA(DL1)	FP/LS	ORIG (cycles per iteration)	STA (ORIG)	REF (cycles per iteration)	STA (REF)	FP (cycles per iteration)	STA (FP)	LS (cycles per iteration)	STA (LS)	DL1 (cycles per iteration)
▼ Loop 703 +	libqmckl.so.0	12.56	RAM bound	1	3.81	SATURATED	1.21	1.36	0.16	95.74	0.75	114.41	0.48	17.54	0.04	112.36	0.46	25.13
○ Bucket 7		66.99	RAM bound	1	3.81	SATURATED	1.21	1.36	0.16	95.74	0.75	114.41	0.48	17.54	0.04	112.36	0.46	25.13
○ Bucket 6		18.57	RAM bound	1	2.75	UNSATURATED	1.19	1.36	0.28	69.18	0.55	72.46	0.48	17.28	0.09	61.54	0.51	25.13
○ Bucket 8		13.42	RAM bound	1	6.12	UNSATURATED	1.20	1.36	0.13	154.00	0.36	143.49	0.79	17.44	0.10	134.72	0.35	25.18

QMCkl: a QMC driven library

1. **Oriented towards performance/portability/productivity but with customized objectives**
2. **Focused and specialized**
3. **Easy to interact with**
4. **Strongly focused on co design**
5. **Using systematically software tools:** multiple versions depending upon target architectures

STATUS:

- A first X86 version available
- GPU and ARM first versions available within 12 to 18 months

l

- 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

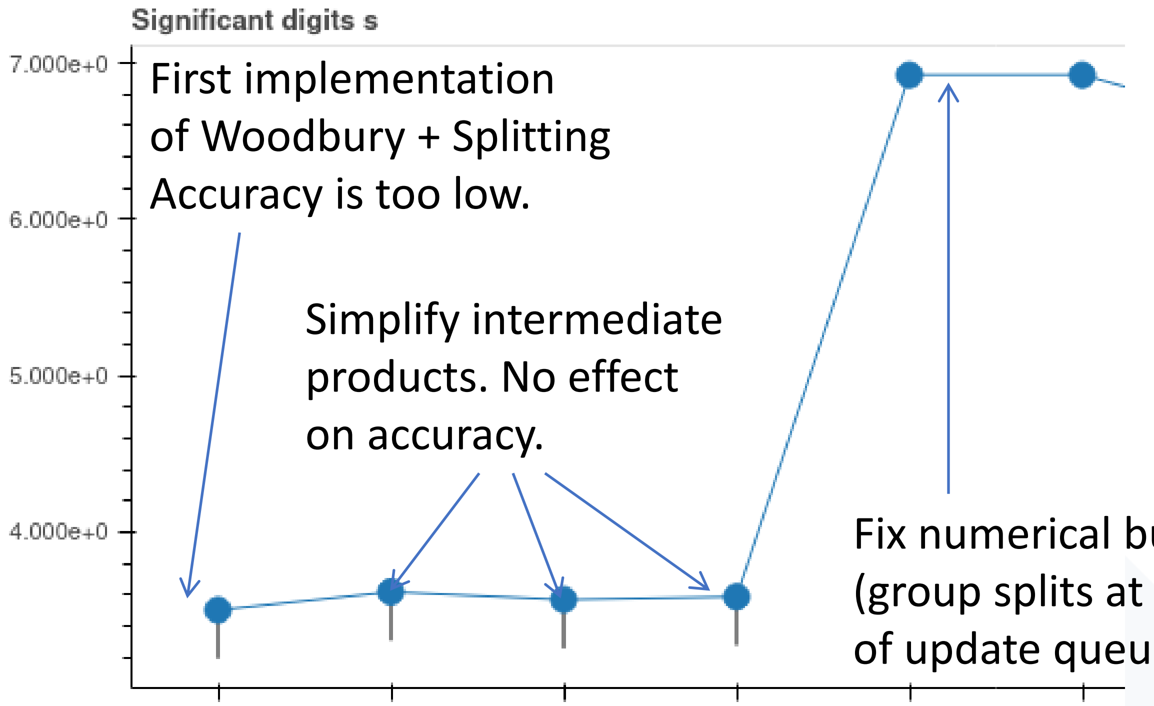
Preserve **numerical accuracy** for new architectures, parallel runtimes, optimizations.
Verificarlo is a tool for assessing the precision of floating point computations.

github.com/verificarlo/verificarlo GPL v3

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

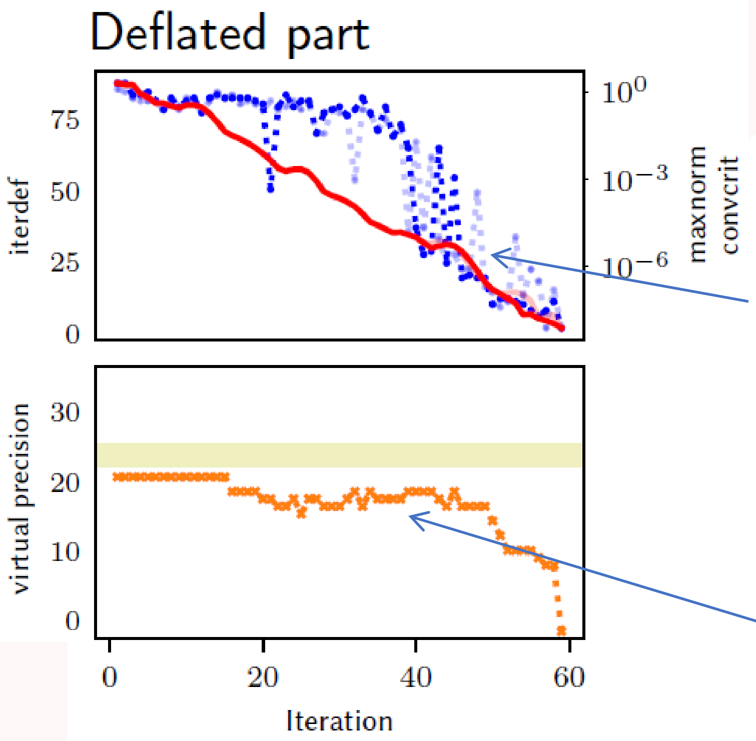
[1] Numerical uncertainty in analytical pipelines leads to impact ul variability in brain networks. Kiar et al. 2021 PLOS ONE.
[2] Study of the effects and benefits of Custom-Precision Mathematical libraries in HPC codes. Brun et al. 2021 IEEE TETC.

Track kernel accuracy during the development process of QMCkl.



Git commits (QMCkl – SMWB kernels)

Harness mixed-precision: 30% speed-up in deflated conjugate gradient (560 cores / YALES2). [3]



Convergence is preserved and…

…the deflated solver runs on **single precision**.

[3] Automatic exploration of reduced floating-point representations in iterative methods. Chatelain et al. Europar'19.

Objectives

1. **Use of automatic generation tools**: from a high level generic ASM generate X86 and ARM versions. This will allow to directly embed low level code (“ASM volatile”).
2. **Use of autotuning tools:** very useful for last mile optimization, explore automatically different code variants and parameters.
3. **Use of advanced performance analysis tools:** monitor not only vectorization ration (% of vector instructions) but also vectorization efficiency (vector width used).
4. **Use of numerical accuracy monitoring tools:** in particular identify code fragments sensitive to accuracy.

A few hundreds of source code lines but restructured for heavy use of dense matrix multiplication operations

The initial equation implemented in CHAMP is:

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

It was rewritten as

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

with

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