Towards automating code generation for (relativistic) many-body electronic structure models

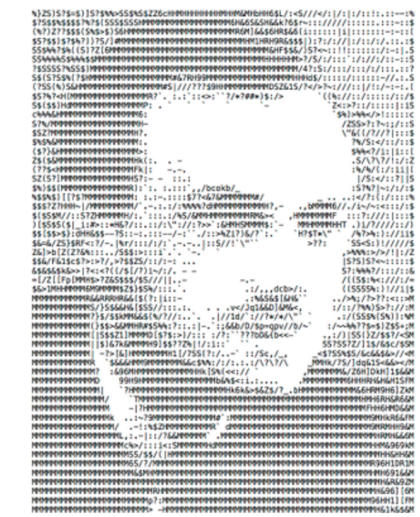
www.cnrs.fr

André Severo Pereira Gomes
andre.gomes@univ-lille.fr

Laboratoire de Physique des Lasers, Atomes et Molécules (PhLAM), CNRS UMR8523
Université de Lille
http://phlam.univ-lille.fr

Acknowledgements

- Collaborators
  - **Dmitry Lyakh** (Oak Ridge), **Loïc Halbert** (Lille)
- Funding and institutional support
- Computing resources



http://diracprogram.org

@DIRACprogram

Tensor contraction engine (TCE) and other tools

- Overview of tools now available for implementing many-body methods for HPC platforms as well as non-relativistic codes can be found here: CR 121, 1203 (2021)
- TCE: Pioneering effort for generating code from many-body theories' equations
  - used e.g. by NWChem (CR 121, 4962 (2021))
  - symbolic manipulator (write equations) and program generator (get code)
    - breakdown of equations into tensor operations

$$A(i,j,k,l) \mathrel{+}= B(i,j,m,n) \times C(m,n,k,l)$$

    - generate (F77) code, after optimizing operations for memory, space etc (like a compiler), manages parallelization (GlobalArrays library)
      - issue: source code is constrained to follow specific parallelization strategy
      - MO basis (no integral-direct code)

PCMT

PhLAM
Physique des Lasers
Atomes et Molécules

What we're doing here…

- … is much more modest, we break the process down into several tools
  - generation of expressions (e.g. CC amplitude equations etc) in terms of binary tensor contractions
    - expressions outputted as instructions in a domain-specific language (SIAL, for "Super Instruction Assembly Language") used in other projects like ACES (JCP 152, 184105 (2020), WIREs 1, 895 (2011))
  - processing of the SIAL instructions
    - which operations, which tensors, order etc
    - handle spatial orbital, spin-orbital formulations
  - output of code of computational kernel (in language of choice) using TAL-SH/ExaTENSOR/exatn primitives
    - library then handles parallelism, GPU usage etc

Processing toolset

- Around 1.6k lines of python

```
andre@utonium ~/D/d/m/src> ls *py
tensor.py          contraction.py       sial.py             codegen_talsh.py
test_tensor.py     test_contraction.py test_parse_sial.py   test_talsh_codegen.py
```

- at top-level, should look like this

```
#!/usr/bin/env python
                                                          test_talsh_codegen.py
# toolset to parse diagen output
# Andre Gomes <andre.gomes@univ-lille.fr>

import codegen_talsh as tcg

def main():

   sial_input = tcg.TALSHcodeGenerator()
   sial_input.read_sial("CCSD.1.fac.u.ormo.txt")
   sial_input.generate_code()
   sial_input.print_code(filename="auto_ccsd.F")

   sial_input = tcg.TALSHcodeGenerator(spinorbital_out=True)
   sial_input.read_sial("CCSD.1.fac.u.ormo.txt")
   sial_input.generate_code()
   sial_input.print_code(filename="auto_ccsd_spinor.F")

main()
```

SIAL sample input fragment

CCSD.1.fac.u.ormo.txt

```
$SIAL CREATE_ARRAY H27(n1a|f1a)
$SIAL CREATE_ARRAY H5(e1be2b|m1bm2b)
$SIAL CREATE_ARRAY H3(e1be2b|f1bf2b)
$SIAL CREATE_ARRAY H11(e1ae1b|m1am1b)
$SIAL CREATE_ARRAY H8(e1ae1b|f1af1b)
$SIAL CREATE_ARRAY H14(e1ae2a|m1am2a)
[…]
$SIAL BARRIER
$SIAL CREATE_ARRAY Z57_1(l1b|d1b)
$SIAL BARRIER
#ORMO        63; Diagram      18; Contraction   1; Tree Level   1; Scaling  2/ 2/ 0/ 0/
0/ 0/ 0.18533025D+08; Result_size  1/ 1/ 0/ 0/ 0/ 0/ 0.43050000D+04
Z57_1(l1b|d1b)+=H24(l1b,l2b|d1b,d2b)*S43(d2b|l2b)*0.5
$SIAL BARRIER
#ORMO        63; Diagram      18; Contraction   2; Tree Level   0; Scaling  1/ 1/ 0/ 0/
0/ 0/ 0.43050000D+04; Result_size  0/ 0/ 0/ 0/ 0/ 0/ 0.10000000D+01
Z48(|)+=S43(d1b|l1b)*Z57_1(l1b|d1b)
$SIAL BARRIER
$SIAL DELETE_ARRAY Z57_1(l1b|d1b)
[…]
```

Sample TAL-SH Fortran code

auto_ccsd_spinor.F

```
subroutine generic_codegen_call(nocc,nvir, H24, S43, H15, S44, H2,\
    H1, H17, H18, H25, H16, H5, H3, H19, H4, H26, Z48, Z49, Z50)
    integer, intent(in) :: nocc
    integer, intent(in) :: nvir
    type(talsh_tensor_t), intent(inout) :: H24    !       OOVV
    type(talsh_tensor_t), intent(inout) :: S43    !       VO
    type(talsh_tensor_t), intent(inout) :: H15    !       OV
[…]
    type(talsh_tensor_t) :: Z57_1    !       OV
    type(talsh_tensor_t) :: Z56_1    !       OO
[…]
    ierr=talsh_tensor_construct(Z57_1, C8, /(nocc,nvir)/, init_val=ZERO)
!   original expression: Z57_1(l1b|d1b)+=H24(l1b,l2b|d1b,d2b)*S43(d2b|l2b)*0.5

ierr=talsh_tensor_contract("Z57_1(l1,d1)+=H24(l1,l2,d1,d2)*S43(d2,l2)",Z57_1,H24,S43
,
                                     scale=(0.5d0,0.0d0))
!   original expression: Z48(|)+=S43(d1b|l1b)*Z57_1(l1b|d1b)
    ierr=talsh_tensor_contract("Z48()+=S43(d1,l1)*Z57_1(l1,d1)",Z48,S43,Z57_1,
                                     scale=(1.0d0,0.0d0))
    ierr=talsh_tensor_destruct(Z57_1)
[…]
```

Current status and short-term goals

- Work in progress
  - finalizing SIAL processing tool
    - can generate code for spatial orbital expressions, still need tweaks to go from that into spin-orbital code
  - identification of “unique” variables that caller code provides (e.g. Fov only, not Fov and Fvo)

```
ierr=talsh_tensor_contract("S(a,i)+=F+(i,a)*K()",s1_tensor,comm_t%fov,one_tensor)
```

    - try to provide code to generate driver routines as well
- Short-term goals
  - computational kernel and driver routines for performing CCD and CCSD pluggable into Exacorr, compare results to current implementation, if ok move on to more complicated models