Moving DBCSR toward new horizon

Alfio Lazzaro

CP2K Development Meeting

July 11, 2017

PASC 2017-2020

- The new PASC project has officially started this month
  - 3 years
  - ~ 2 FTE (hopefully we will hire a PostDoc soon)
- There were 17 projects, only 10 accepted
  - http://www.pasc-ch.org/projects/2017-2020/
- CSCS will provide support with Software Engineers
  - Future Systems Group, head by Joost (16 people in the group)
  - Not clear how we will collaborate with them

Computing equilibria in heterogeneous agent macro models on contemporary HPC platforms

Computing equilibria in heterogeneous agent macro models on contemporary HPC platforms; PI: Felix Kübler (University of Zurich)

ENIAC

ENIAC – Enabling the ICON model on heterogeneous architectures; PI: Ulrike Lohmann (ETH Zurich)

FASTER

FASTER - Forecasting and Assessing Seismicity and Thermal Evolution in geothermal Reservoirs; PI: Thomas Driesner (ETH Zurich)

HPC-PREDICT

HPC-PREDICT – High-Performance Computing for the Prognosis of Adverse Aortic Events; PI: Dominik Obrist (University of Bern)

PASCHA

PASCHA – Portability And Scalability of COSMO on Heterogeneous Architectures; PI: Torsten Höfler (ETH Zurich)

Salvus

Salvus; PI: Andreas Fichtner (ETH Zurich)

SIRIUS

Development and optimization of the domain-specific library SIRIUS for electronic-structure calculations, and integration within the Quantum-ESPRESSO distribution; PI: Nicola Marzari (EPFL)

Sparse tensor linear algebra library

Sparse Tensor Linear Algebra Library; PI: Jürg Hutter (University of Zurich)

SPH-EXA

SPH-EXA: Optimizing Smooth Particle Hydrodynamics for Exascale Computing; PI: Florina Ciorba (University of Basel), Lucio Mayer (University of Zurich)

Virtual Physiological Blood

Virtual Physiological Blood: an HPC framework for blood flow simulations in vasculature and in medical devices; PI: Petros Koumoutsakos (ETH Zurich)

PASC proposal

- Three main areas
  1. Extension to Tensor algebra
     - Building the tool on top of DBCSR, mapping operations to DBCSR matrices
     - Patrick has already started to work on it (under dbcsr_tensor)
  2. Generate GPU kernels on-the-fly (Just-in-time compilation)
     - Currently we generate all kernels with an autotuning procedure, collected in libcusmm
       - Andreas is working on updating the kernels for the new GPU on Daint
       - Pretty big library with 2349 kernels
     - If a kernel is not in libcusmm, the library will call the CPU implementation, with lost of performance
     - We want to generate a relative small set of kernels and use them in a “machine learning”-box to quickly generate any other kernel
  3. Improving load-balancing
     - Main effect from MPI communications, small effect from computation part
     - Several possible approaches, no clear if we will gain in overall performance at the end

Referees' replies

- 5 reviews
  - All agreed that it is an interesting project
- Some "weakness" comments
  - The work may result in only incremental improvement and may not be adopted in other domains. Indeed, similar to the chemistry-based effort here, other domains also develop their packages using tensors when needed.
  - The reliance on DBCSR reduces the global impact of this work.
  - How NWChem could use the proposed library while as far as I know NWChem have its own MPI layer and the tensor computation happen locally. How such integration could happen?
  - There is little evidence in this proposal that the software produced by this project will have a significant impact.

Lesson learnt

- **We need to disseminate DBCSR!**
  - Integration and comparison with other projects
  - **Improve documentation**
  - Improve interface (C, Python)

- Include DBCSR in distributions, e.g.
  - ELSI (already mentioned in the PASC proposal)
  - E-CAM Electronic Structure Module Library

Highest priority!

DBCSR homepage

DBCSR API Documentation

Overview of CP2K API

All Modules

Packages:

[root]
acc
aobasis
arnoldi
base
common
dbcsr
dbcsr/base
dbcsr/block
dbcsr/data
dbcsr/dist
dbcsr/mm
dbcsr/ops
dbcsr_tensor
dbcsrx
emd
eri_mme
fm

All Modules:

acc_device
acc_devmem
acc_event
acc_hostmem
acc_stream
admm_dm_methods
admm_dm_types
admm_methods
admm_types
admm_utils
ai_angmom
ai_contraction
ai_contraction_sphi
ai_coulomb
ai_coulomb_test
ai_derivatives
ai_elec_field
ai_eri_debug
ai_fermi_contact
ai_kinetic
ai_moments
ai_onecenter

Documentation for module `dbcsr_api`

[apidoc]

This is the start of a dbcsr_api, all publically needed functions are exported here. The others remain private to the library. Currently, this is the CP2K used set. Ultimately, a reduced subset and well defined api will remain, possibly grouped in to standard and expert api. Currently, this is work in progress.

source: dbcsr_api.F

Introduction:

[Create intro]

Forwarded symbols:

from dbcsr_config:

dbcsr_get_default_config → dbcsr_config::dbcsr_get_default_config
...

dbcsr_print_config → dbcsr_config::dbcsr_print_config
Prints configuration for DBCSR

dbcsr_set_config → dbcsr_config::dbcsr_set_config
...

from dbcsr_csr_conversions:

csr_dbcsr_blkrow_dist → dbcsr_csr_conversions::csr_dbcsr_blkrow_dist
...

csr_destroy → dbcsr_csr_conversions::csr_destroy
destroy a CSR matrix

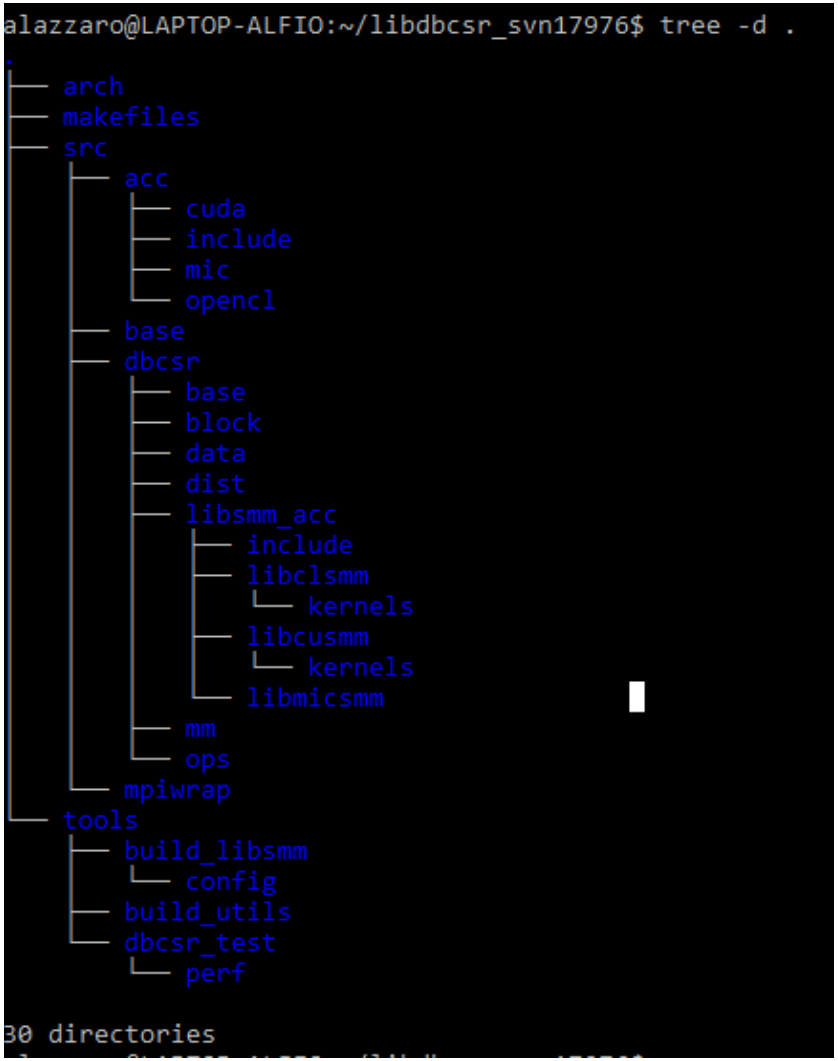
csr_eqrow_ceil_dist → dbcsr_csr_conversions::csr_eqrow_ceil_dist
...

csr_eqrow_floor_dist → dbcsr_csr_conversions::csr_eqrow_floor_dist
...

Loading [Contrib]/a11y/accessibility-menu.js

csr_print_sparsity → dbcsr_csr_conversions::csr_print_sparsity

DBCSR tar file



Extract from CP2K

- **Same repository**
- **Same makefile**
- **Same Arch files**
- **mpiwrap's**
- **Same conventions**

DBCSR strength points

- MPI + OpenMP parallelization
  - Able to parallelize on multi-nodes and multi/many-cores
  - Intel Libxsmm provides fast execution on CPU and it is fully supported (Hans et al.)
- Work well for sparse to dense systems
- Apply on-the-fly filtering
  - Reduce computation, preserve sparsity
- Efficient CUDA backend
- Able to stress the network/CPU/GPU
  - Can be used as benchmark for testing the hardware
- APIs for several high-level functionalities

Promote DBCSR to other codes: what's missing?

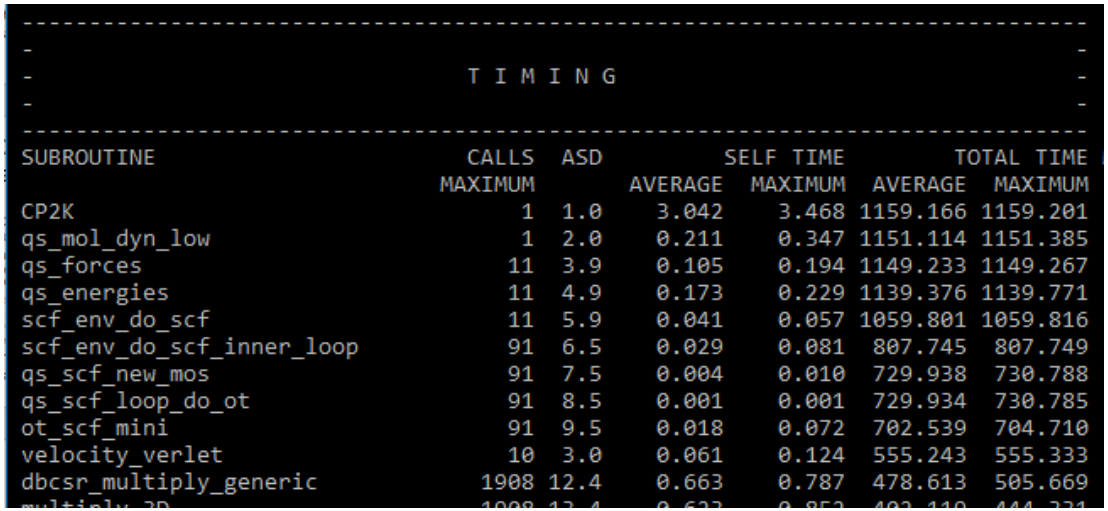
- Logo ➔ DONE
- There is no DBCSR related version
  - It is based on CP2K version
- There is no standalone repository
  - Developers cannot contribute, unless they have access to CP2K repository
- There is no a test suite, but some unit tests
  - Heavily tested in CP2K, but what if we add a functionality used by another code?
  - Unit tests doesn't check all cases
- Integration with other libraries
- Real documentation

Proposal

- Standalone DBCSR repository (Github)
  - Introducing a version number (XX.YY.ZZ)
- New compilation ecosystem (makefile, …) and tools (prettified, fypp)
  - I know, it can be a duplication of work with respect to CP2K
- Extend current unit tests and include tests taken from CP2K
  - We dump some matrices and use them as DBCSR input
  - We can use the entire CP2K to test DBCSR
- Make CP2K depending on DBCSR as an external library
  - CP2K will use specific DBCSR versions
  - Therefore DBCSR and CP2K can evolve autonomously
- Provide C interface for some functions
  - In total DBCSR API counts 124 functions

From CP2K to DBCSR to CP2K (1)

- Timing output from CP2K is not in DBCSR
  - It is switched off
  - Actually the code for timings it is not included at all



```
-------------------------------------------------------------------------------
-                                                                             -
-                                T I M I N G                                  -
-                                                                             -
-------------------------------------------------------------------------------
SUBROUTINE                   CALLS  ASD         SELF TIME        TOTAL TIME
                            MAXIMUM       AVERAGE  MAXIMUM  AVERAGE  MAXIMUM
CP2K                              1  1.0    3.042    3.468 1159.166 1159.201
qs_mol_dyn_low                    1  2.0    0.211    0.347 1151.114 1151.385
qs_forces                        11  3.9    0.105    0.194 1149.233 1149.267
qs_energies                      11  4.9    0.173    0.229 1139.376 1139.771
scf_env_do_scf                   11  5.9    0.041    0.057 1059.801 1059.816
scf_env_do_scf_inner_loop        91  6.5    0.029    0.081  807.745  807.749
qs_scf_new_mos                   91  7.5    0.004    0.010  729.938  730.788
qs_scf_loop_do_ot                91  8.5    0.001    0.001  729.934  730.785
ot_scf_mini                      91  9.5    0.018    0.072  702.539  704.710
velocity_verlet                  10  3.0    0.061    0.124  555.243  555.333
dbcsr_multiply_generic         1908 12.4    0.663    0.787  478.613  505.669
```

  - DBCSR functions will be not included in the CP2K list
    - There will be a specific DBCSR list of timings

From CP2K to DBCSR to CP2K (2)

- Standardize the output and error checking so that it doesn't conflict with CP2K
- MPI wrappers and some other functionalities require special attention
  - Avoid conflicts with the CP2K code

Documentation

- My plan is to write a document with two levels of explanations
  - For users: details on how to use the library
  - For developers: further details on how the library is implemented
  - Include examples
- Improve code comments
- A lot of code to review
  - I know, it will be a nightmare...
  - Hopefully I will start to circulate a first draft by September

Other ongoing projects

- Andreas is working on reoptimizing the GPU kernels
  - Special optimization when a single kernel is used
- Finalizing RMA algorithm
  - I just need to write a wiki page with explanations on how to use it
- KNL evaluation
  - We have to finish a paper for the ParCO conference by the end of the month (presentation was already accepted)
- Use OpenMP tasks, as suggested by Iain et al
  - Threads and more threads
  - Great potential in improving performance
    - Reduce overhead by making dynamic load-balancing
    - Avoid a lot of implicit synchronizations

Conclusion and timeline

- DBCSR is doing well
- Short term (~2 months)
  - GPU kernel optimization (Andreas)
  - KNL evaluation
  - RMA algorithm
  - Collaboration with Intel (Hans)
- Medium term (~6 months)
  - OpenMP task implementation (Iain et al.)
  - Tensor prototype by Patrick
  - DBCSR documentation
  - Standalone DBCSR repository
  - Ask other people to use DBCSR

PASC proposal timeline

- Long term (2-3 years)
  1. Extension to Tensor algebra
     - 2 years
  2. Generate GPU kernels on-the-fly (Just-in-time compilation)
     - 1 year
  3. Improving load-balancing
     - 1 year


I SAW YOUR EMAIL ABOUT DESTROYING THE COMPANY.
HUH?
THE ONLY EMAIL I SENT YOU WAS MY STRATEGY FOR THE COMING YEAR.
WELL, MAYBE I READ IT TOO FAST.
Dilbert.com @ScottAdamsSays
12-22-16 © 2016 Scott Adams, Inc. /Dist. by Universal Uclick