The DBCSR Library

Shoshana Jakobovits

24 June 2020
The DBCSR Library

- **Distributed**
  - MPI parallelization is based on the Cannon algorithm
  - On node with OpenMP

- **Block Compressed Sparse Row**
  - Block-sparse, where blocks correspond to atoms

- **Sparse matrix-matrix multiplication library (SpGEMM)**
  - As well as other linear algebra operations
  - And: tensor contractions framework
The DBCSR Library

- Written in Fortran
- Types supported: double
- GPU accelerated for NVIDIA & AMD via CUDA & HIP
- First written as a part of CP2K, but now released as standalone
DBCSR has a Fortran API as well as a C API
Code snippets: initialize the library

```fortran
! initialize libdbcsr
CALL dbsr_init_lib(MPI_COMM_WORLD)

!
! the matrix will contain nblkrows_total row blocks and nblkcols_total column blocks
nblkrows_total = 4
nblkcols_total = 4

!
! set the block size for each row and column
ALLOCATE (row_blk_sizes(nblkrows_total), col_blk_sizes(nblkcols_total))
row_blk_sizes(:) = 2
col_blk_sizes(:) = 2
```
! set the row and column distributions (here the distribution is set randomly)
CALL random_dist(row_dist, nblkrows_total, npdims(1))
CALL random_dist(col_dist, nblkcols_total, npdims(2))

! set the dbcsr distribution object
CALL dbcsr_distribution_new(dist, group=group, row_dist=row_dist, col_dist=col_dist, reuse_arrays=.TRUE.)
! create the dbcsr matrices, i.e. a double precision non symmetric matrix
! with nblkrows_total x nblkcols_total blocks and
! sizes "sum(row_blk_sizes)" x "sum(col_blk_sizes)", distributed as
! specified by the dist object
CALL dbcsr_create(matrix=matrix_a, &
    name="this is my matrix a", &
    dist=dist, &
    matrix_type=dbcsr_type_no_symmetry, &
    row_blk_size=row_blk_sizes, &
    col_blk_size=col_blk_sizes, &
    data_type=dbcsr_type_real_8)
! set up the a matrix
CALL dbcsr_distribution_get(dist, mynode=mynode)
ALLOCATE (values(max_nze))
DO row = 1, dbcsr_nblkrows_total(matrix_a)
   DO col = MAX(row - 1, 1), MIN(row + 1, dbcsr_nblkcols_total(matrix_a))
      row_s = row; col_s = col
      CALL dbcsr_get_stored_coordinates(matrix_a, row_s, col_s, node_holds_blk)
      IF (node_holds_blk .EQ. mynode) THEN
         nze = row_blk_sizes(row_s)*col_blk_sizes(col_s)
         CALL RANDOM_NUMBER(values(1:nze))
         CALL dbcsr_put_block(matrix_a, row_s, col_s, values(1:nze))
      ENDIF
   ENDDO
ENDDO
DEALLOCATE (values)
Code snippets: multiply matrices

```
! finalize the dbcsr matrices
CALL dbcsr_finalize(matrix_a)
! b, c ...

!
! multiply the matrices
CALL dbcsr_multiply('N', 'N', 1.0D0, matrix_a, matrix_b, 0.0D0, matrix_c)
!
!
! print the matrices
CALL dbcsr_print(matrix_a)
! b, c ...
!
! release the matrices
CALL dbcsr_release(matrix_a)
! b, c ...
```
Installation & Requirements

Requirements

- Cmake and GNUmake, Ninja
- A BLAS + LAPACK implementation
- (Optional: libxsmm)

Build

```
cmake .. \
-DUSE_MPI=<ON|OFF> -DUSE_OPENMP=<ON|OFF>
-DUSE_SMM=<blas|libxsmm>
-DUSE_CUDA=<OFF|ON> -DUSE_HIP=<OFF|ON>
-DWITH_GPU=<P100|K20X|K40|K80|V100|Mi50>
```
Natural use case: Electronic Structure

Application-specific sparsity patterns

DBCSR is based on blocked structure

- Non-zero elements are small dense blocks
  - typically 13x13, 23x23, ...

- Take full advantage of the block structured sparse nature of the matrices

- Each block corresponds to the interaction between two atoms
DBCSR Software Structure

- MPI Parallelization
  Data-exchange layout
- Traversal
  Cache Optimization
- Generation
  Batches generation
- Scheduler
  CPU/GPU Load balancing
- Distributed

- Host Driver
  - BLAS
  - LIBXSMM

- Device Driver
  - cu/hip-BLAS
  - LIBSMM_ACC

- Node
libsmm_acc: library for small matrix-matrix multiplications on accelerators
libsmm_acc: parametrized CUDA kernels

- grouping
- number of CUDA threads
- minblocks
- tile_m, tile_n
- P_a, P_b
  +
- algorithm

- Performance is difficult to model
- There are tradeoffs and interactions between parameters
libsmm_acc: parametrized CUDA kernels

We use auto-tuning data, augmented with kernel and hardware derived features to train a Machine Learning model that can accurately predict optimal parameters for new matrix-matrix multiplication dimensions.
Linear SCF scaling

Linear Scaling Self-Consistent Field Calculations with Millions of Atoms in the Condensed Phase
Joost VandeVondele, Urban Borštnik, and Jürg Hutter
Journal of Chemical Theory and Computation 2012 8 (10), 3565-3573
DOI: 10.1021/ct200897x
GPU back end performance

<table>
<thead>
<tr>
<th>GPU</th>
<th>Peak FP64 Perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Mi50</td>
<td>6.6 Tflop/s</td>
</tr>
<tr>
<td>NV V100</td>
<td>7 Tflop/s</td>
</tr>
<tr>
<td>NV P100</td>
<td>4.7 Tflop/s</td>
</tr>
</tbody>
</table>
Use DBCSR

- Available as a standalone
  - Github: actively developed
  - Documentation
  - Easy to use & install

https://github.com/cp2k/dbcsr

- Also delivered automatically with any CP2K installation, so if you’re running CP2K, you’re also running DBCSR (perhaps without knowing it)

Number of lines of code ~75k
Languages: 82% Fortran, 8% Python, 5% C++
License: GPL-2.0
THANKS

Follow us on:

LinkedIn: company/max-centre/
Twitter: @max_center2
Website: http://www.max-centre.eu/