DBCSR: A Sparse Matrix Multiplication Library for Electronic Structure Codes

Andreas Glössl1, Alfio Lazzaro1, Patrick Seewald2, Ole Schütte1, Hans Pabst1, and Jürg Hutter2
1Department of Chemistry, University of Zurich, Winterthurerstr. 190, CH-8057 Zurich
2EMP, Überlandstrasse 129, CH-8600 Dübendorf
*email: swissacel@acm.org

1 Introduction
Sparse matrix-matrix multiplication is an essential building block for electronic structure theory calculations. For this task, the sparse matrix library DBCSR has been developed1. Its multi-layered structure automatically takes care of and optimizes several computational aspects like parallelism (MPI, OpenMP, GPU), data (cache) locality and on-the-fly filtering We introduce a framework for sparse tensor linear algebra, which enables low-scaling electronic structure methods beyond density functional theory. We also present performance results for the backends, namely LIBXSMM and LIBCUSMM2. Finally, we show some preliminary performance results when running DBCSR on a supercomputer equipped with Intel Xeon Phi processor, code name Knights Landing (KNL).

2 DBCSR Overview
- Typical occupancy 0.1%–100%
- Matrices are stored in a blocked compressed sparse row (CSR) format
- Non-zero elements are small dense blocks, indexed by the CSR index
  - Typical block sizes in the range of 1–50
- Small matrix multiplications (SMM) are organized in "matrix stacks", and special libraries are deployed for computing the SMM stacks
  - Cluster Layer: MPI/OpenMP load-balancing and block distribution
  - Multilevel Layer: Optimize memory access, cache-oblivious algorithms
  - CSR Layer: Indexing and create/sort/filter stacks
  - Scheduler and Driver Layers: Stack processing
1. Permutation of rows and columns, randomly distributed, to obtain good load balance
2. Distribution over a two-dimensional grid of P processes, e.g. 1 × 4 grid
   - Static decomposition
3. Intra-node communications based on a communication-reducing algorithm3
   - Implementation based on 2.5D algorithm4
   - MPI communications based on One-sided MPI
4. Local node execution of stacks in parallel by means of OpenMP threads
   - Static assignment of multiplications to threads
   - Batch execution of the multiplications on the CPU (LIBXSMM) and GPU (LIBCUSMM)

3 Tensor Framework
3.1 Example: Fast Hartree-Fock exchange
Applications of our sparse tensor framework include cubic scaling RPA1 and a similar approach to fast, quadratic scaling Hartree-Fock exchange (HF2). As any algorithm consisting of subsequent tensor contractions, the HF2 algorithm can be represented equivalently in terms of tensors or matrices. The tensor model is based on tensor contractions (TC). The matrix model is based on matrix multiplication (MM) and matrix rank conversion (MC) steps.

3.2 Design: Tensor view on matrix data
A light-weight tensor interface to DBCSR bridges the gap between the tensor model and the matrix model. While tensor contraction is fully based on DBCSR matrix multiplication, the tensor interface hides the matrix model. Thus algorithms involving sparse tensors can be directly implemented in the tensor model. In order to preserve data locality in terms of atomic blocks, tensors are mapped block-wise to DBCSR matrices (blocked column-major order). The DBCSR tensor framework is generic in the sense that arbitrary contractions between tensors of arbitrary ranks and arbitrary data types are supported.

3.3 Tensor contraction
A tensor contraction (TC) is a combination of matrix conversion (MC) steps and one matrix-matrix multiplication (MM): Only the tensor representation is visible to the outside and consistent matrix layouts are automatically chosen.

3.4 Matrix conversion
Matrix conversion (MC) is the conversion between arbitrary 2d representations of the same tensor and involves a complete redistribution of tensor blocks and local reshape of matrix data.

3.5 Rectangular matrix-matrix multiplication
Traditional algorithms for parallel matrix-matrix multiplication (2D algorithm) perform well only for square matrices. For tensor contractions, we need a communication-avoiding algorithm for rectangular matrices.7

4 CUDA-Kernels
CUDA-Kernel parameters depend on hardware specifications (number of MP, registers, and size of memories). However, the best launch-, tile- and block-sizes are determined by a benchmark for each individual kernel using an empirically found heuristic.
For the transition from K20x to P100:
- No better heuristic has been found
- Max. performance increased from 45% to 65% of peak.

5 Performance results on KNL system
- Preliminary results: first tests on a KNL system, no specific code optimizations
- Configurations
  1. Cray XC40 KNL "Grand Tavé" at CSCS
  2. Cray XC50 GPU-partial "Daint" at CSCS
- Tests performed within the CP2K package with application benchmarks

6 Summary/Outlook
DBCSR is freely available at http://dbcsr.cp2k.org as stand-alone, general purpose, sparse matrix multiplication library including sample code.
Future development on DBCSR under the project Sparse Tensor Linear Algebra Library funded by PASC 2017–2020.
Improving DBCSR as a library to facilitate usage in electronic structure codes beyond CP2K (collaboration with ELSI project), numerical libraries and other scientific domains.

References
[1] E. Solomonik and J. Demmel


Figure 1: Vlasov simulation performed with Vlasov code using DBC-CDM on a 16-k node, 1024-core P100 system, with up to 2 Teraflops of peak performance.