PASC17 Lugano Switzerland 26–28 June 2017

University of Zurich[UZH]

DBCSR: A Sparse Matrix Multiplication Library for Electronic Structure Codes

Andreas Glöss[1], Alfio Lazzaro[1], Patrick Seewald[1], Ole Schütt[2], Hans Pabst[3], and Jürg Hutter[1]

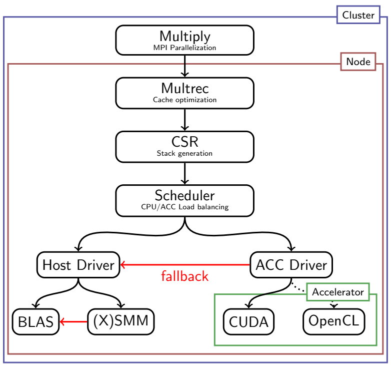
[1]Department of Chemistry, University of Zurich, Winterthurerstr. 190, CH-8057 Zurich
[2]EMPA, Überlandstrasse 129, CH-8600 Dübendorf
[3]Intel Semiconductor AG, Switzerland

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 developed[1]. 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 LIBCUSMM[2;3]. Finally, we show some preliminary performance results when running DBCSR on a supercomputer equipped with Intel Xeon Phi processor, code name Knights Landing (KNL).

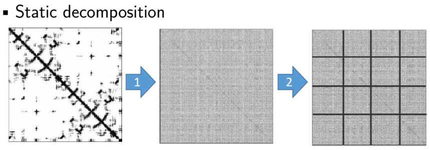
2 DBCSR Overview

- 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 blocks 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
- Multrec Layer: Optimize memory access, cache-oblivious algorithm
- 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.* 4×4 grid
   - Static decomposition



3. Intra-node communications based on a communication-reducing algorithm[4]
   - Implementation based on 2.5D algorithm[5]
   - 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 RPA[6] and a similar approach to fast, quadratic scaling Hartree-Fock exchange (HFX).
As any algorithm consisting of subsequent tensor contractions, the HFX 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 conversion (MC)** steps.

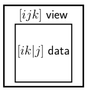
HFX in the tensor model

TC: $[Q\lambda\mu] = [\mu\lambda P][PQ]$
for each SCF step **do**
  TC: $[Q\lambda\nu] = [Q\sigma\nu][\lambda\sigma]$
  TC: $[\mu\nu] = [Q\lambda\mu][Q\lambda\nu]$
end for

HFX in the matrix model

MM: $[\mu\lambda|Q] = [\mu\lambda|P][P|Q]$
for each SCF step **do**
  MM: $[Q\nu|\lambda] = [Q\nu|\sigma][\sigma|\lambda]$
  MC: $[\mu|\lambda Q] = [\mu\lambda|Q]$
  MC: $[\nu|\lambda Q] = [Q\nu|\lambda]$
  MM: $[\mu|\nu] = [\mu|\lambda Q][\nu|\lambda Q]^T$
end for

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

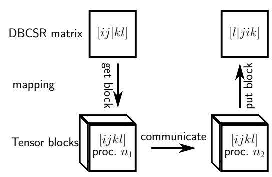
A tensor contraction (TC) is a combination of matrix conversion (MC) steps and one matrix-matrix multiplication (MM):

$[ijk]\times$	$[iklm] =$	$[jlm]$	Tensor model
$[ij\|k]$	$[kl\|im]$	$[mj\|l]$	
$\downarrow$ MC	$\downarrow$ MC	$\uparrow$ MC	Matrix model
$[j\|ik]\times$	$[ik\|lm] \xrightarrow{\text{MM}}$	$[j\|lm]$	

Only the tensor representation is visible to the outside and consistent matrix layouts are automatically chosen.

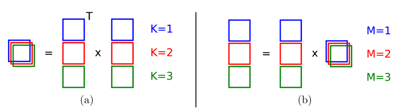
3.4 Matrix conversion

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 (2.5D algorithm[5]) perform well only for square matrices. For tensor contractions, we need a communication-avoiding algorithm for rectangular matrices:[7]



$$C_{mn} = \sum_K \left(\sum_{k \in K} A_{mk} B_{kn} \right) = \sum_K [C_K]_{mn}$$

group-local MM — reduction

$$[C_{mn}]_{m\in M} = \sum_k [A_{mk}]_{m\in M} B_{kn}$$

group-local MM

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.

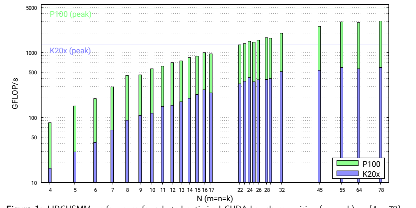


Figure 1: LIBCUSMM performance for selected optimized CUDA-kernels comprising (m=n=k) = {4,...,78} matrices. The FLOP-rates, as obtained from individual kernel launches in a mini-app, are shown as blue (K20x) and green (P100) bars.

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
     - 64 cores Intel Xeon Phi CPU 7230 @ 1.30GHz
     - MCDRAM in cache mode, QUADRANT cluster mode
  2. Cray XC50 GPU-partition "Daint" at CSCS
     - 12 cores Intel Xeon CPU E5-2690 v3 @ 2.60GHz and NVIDIA Tesla P100
- Tests performed within the CP2K package with application benchmarks[8]

	H2O-DFT-LS	S-E	AMORPH
Block sizes $(m \times n)$	23×23	6×6	5×13
# Rows/columns	158,976	1,119,744	133,214
Occupancy range (%)	$7-15$	$(4-6)\,10^{-2}$	$5-70$
# Multiplications	193	618	94
DBCSR FLOPs $(\times 10^{15})$	4.038	0.074	1.680

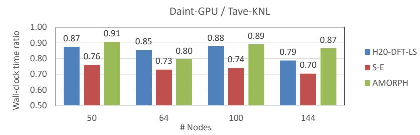


Figure 2: Relative performance of Tavé KNL versus Daint-GPU: values less than 1 mean that Daint-GPU is faster than the Tavé KNL. Averages: 0.85 for H2O-DFT-LS, 0.73 for S-E, 0.86 for AMORPH. Fluctuations are below 4%.

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*[9] project), numerical libraries and other scientific domains.

References

[1] U. Borštnik *et al.*, *The distributed block-compressed sparse row library*, Parallel Computing, **2014**, *40*(5-6): 47-58

[2] A. Heinecke *et al.*, *LIBXSMM: Accelerating Small Matrix Multiplications by Runtime Code Generation*, in proceedings of the SC'16, **2016**

[3] O. Schütt *et al.*, *GPU Accelerated Sparse Matrix Matrix Multiplication for Linear Scaling DFT*, J. Wiley & Sons **2015**

[4] A. Lazzaro *et al.*, *Increasing the Efficiency of Sparse Matrix-Matrix Multiplication with a 2.5D Algorithm and One-Sided MPI*, in proceedings of the PASC'17, **2017**

[5] E. Solomonik and J. Demmel, *Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms*, European Conference on Parallel Processing, Springer, **2011**

[6] J. Wilhelm *et al.*, *Large-scale cubic-scaling RPA correlation energy calculations using a Gaussian basis.*, Journal of Chemical Theory and Computation, **2016**

[7] J. Demmel *et al.*, *Communication-optimal parallel recursive rectangular matrix multiplication*, 27th International Symposium on Parallel & Distributed Processing, **2013**: 261-272.

[8] The CP2K developers group, CP2K is freely available from: http://www.cp2k.org/, **2015**

[9] V. Yu *et al.*, *ELSI: A Unified Software Interface for Kohn-Sham Electronic Structure Solvers.*, arXiv preprint arXiv:1705.11191, **2017**.