1 Introduction

The DBCSR library provides an optimized implementation for carrying dense and sparse matrix-matrix multiplication. 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. The library is implemented in Fortran 2003 and it is available on a GitHub repository.

Here, we introduce the library implementation. Then, we report on the latest performance optimization for improving the CUDA implementation by means of a novel algorithm for the work-scheduling of the multiplication of the matrix blocks. Finally, we report on a coverage analysis when using compilers from different vendors for building the library and testing it within the CP2K quantum chemistry and solid state physics software package.

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: OpenMP/Offload load balancing and block distribution
- Multirec Layer: Optimizes 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. $4 \times 4$ grid
   - Static decomposition

3. Intra-node communications based on a communication-reducing algorithm
- Implementation based on 2.5D algorithm
- 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 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 NVIDIA K20x to NVIDIA P100 GPU cards we did the following optimizations:
- Reduce the ‘param_stack’ from 7 to 3 elements per stack entry.
- Remove ‘careful’ and ‘runs’ in favor of ‘stack_size’.
- Use ‘_ldg()’ to read directly into shared memory.
- Eliminate branches by using sufficiently large shared buffers.
- Only ‘synchronize()’ if buffer/work is shared between warps of a threadblock.
- PTX and SASS analysis for some selected kernel sizes and kernel types.
- Reduce number of ‘if’, with the drawback of having more lines of code.
- Reoptimize kernels for new/better parameter sets (1455 kernels).

4 Compilers Coverage Testing

We compile DBCSR within the CP2K framework with different compilers for testing their correctness. Indeed, while CP2K and DBCSR adhere to the Fortran 2003 standard, not all compilers (or compiler versions) are able to build them. We compile DBCSR within the CP2K framework with different compilers for testing their correctness. Indeed, while CP2K and DBCSR adhere to the Fortran 2003 standard, not all compilers (or compiler versions) are able to build them. While the lower curve shows the final results obtained with re-optimized P100-Kernel code, the upper curve shows the results obtained by relaxing parameter constraints, while the upper curve shows the final results obtained with re-optimized P100-Kernel code.

5 Fork me on GitHub!

https://github.com/cp2k/dbcsr

- Open Source licensed: GPL v2+
- 18 years of development history, 1500 commits, 25 contributors
- Forked out from CP2K Subversion
- Using git filter and git filter-branch
- Completely standalone version
- Regularly synced to CP2K
- Continuous Integration with Travis and Codecov

6 Summary/Outlook

DBCSR is a stand-alone, general purpose, sparse matrix multiplication library including sample code and freely available at https://github.com/cp2k/dbcsr. Join us on Github!

- It is continuously developed for extending its API (including a C/C++ API) and improving its performance.
- The library is extensively tested within the CP2K framework by using different configurations and compilers.
- 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 ELS² project), numerical libraries and other scientific domains.

References

[2] The 2.5D approach is documented in: http://www.sci.utah.edu
in accordance with the FPC C17, 2007
[4] S. Vetter et al., "Communication优化ed parallel 2.5D matrix multiplication and 2.5D Strassen algorithms.,

50 500 1000 4 5 6 7 8 9 10 11 12 13 14 15 16 17 22 24 26 28 32 45 55 64 78 96
P100 (peak) K20x (peak)
GFLOP/s
N (m=n=k)
P100*K20x@P100
K20x

Figure 1: Detailed view on CUDA-Kernel performance for individually labeled Kernel-Flavours comprising (n,m,k) and matrix at P100 (green), different parameter sets. Stars generated with K20x kernels are drawn at 4k GFLOP/s. The lower curve represents results obtained by relaxing parameter constraints, while the upper curve shows the final results obtained with re-optimized P100-Kernel code. The bars represent results obtained by relaxing parameter constraints, while the upper curve shows the final results obtained with re-optimized P100-Kernel code.