University of Zurich[UZH]

Increasing the Efficiency of Sparse Matrix-Matrix Multiplication with a 2.5D Algorithm and One-Sided MPI

Alfio Lazzaro[1], Joost VandeVondele[2], Jürg Hutter[1], Ole Schütt[3]

[1] University of Zürich, [2] CSCS, [3] EMPA

PASC17 Lugano Switzerland 26-28 june 2017

PASC 2017, Lugano (CH)

June 26th, 2017

Overview

Increasing the Efficiency of Sparse Matrix-Matrix Multiplication with a 2.5D Algorithm and One-Sided MPI

Overview

Increasing the Efficiency Sparse Matrix-Matrix Multiplication 2.5D Algorithm One-Sided MPI

Overview

- Sparse Matrix-Matrix Multiplication
  - Focus on Linear Scaling Density Functional Theory
  - Introducing Distributed Block Compressed Sparse Row (DBCSR) library
- New 2.5D Algorithm
  - Comparison with previous DBCSR Cannon's algorithm
- One-sided MPI implementation
- Performance
- Conclusion and outlook

Overview

- Sparse Matrix-Matrix Multiplication
  - Focus on Linear Scaling Density Functional Theory
  - Introducing Distributed Block Compressed Sparse Row (DBCSR) library
- New 2.5D Algorithm
  - Comparison with previous DBCSR Cannon's algorithm
- One-sided MPI implementation
- Performance
- Conclusion and outlook

Sparse Matrix-Matrix Multiplication (SpGEMM)

- Applications in a wide range of domains, such as
  - Finite element simulations based on domain decomposition
  - Computational fluid dynamics
  - Climate simulation
  - Big Data
  - Electronic structure

Application field of this presentation

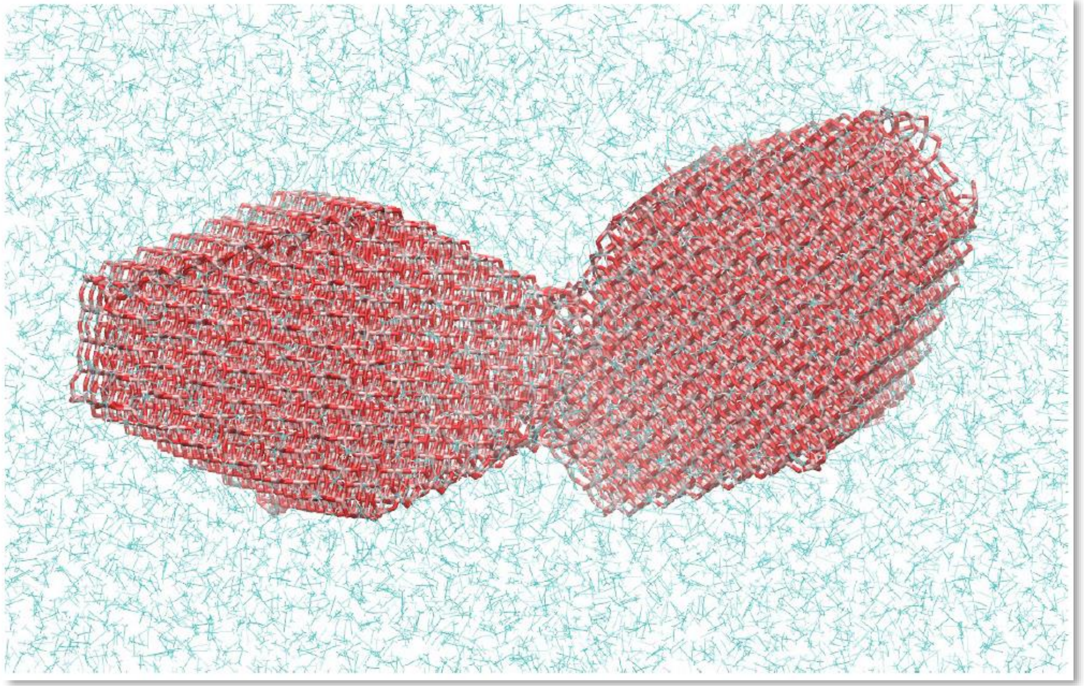
Distributed SpGEMM challenges

- Parallel SpGEMM is an irregular computation (load balance) whose performance is communication bound
  1. Improve performance if there is *a priori knowledge* about the input or output matrices sparsity structure
  2. In the *general case* a priori knowledge of the input and output matrix sparsity is unknown

Case study of this presentation

Application Field: Electronic Structure

- Simulation of nanoparticles, electronic devices, macromolecules, disordered systems, a small virus
- Simulation based on Density Functional Theory (DFT)



Aggregated nanoparticles in explicit solution (77,538 atoms). Relevant for 3rd generation solar cells.
Run in 2014 with **CP2K** on the CSCS Piz Daint supercomputer (Cray XC30, 5272 hybrid compute nodes, 7.8PF) at approx. 122s per step (requires thousands steps)

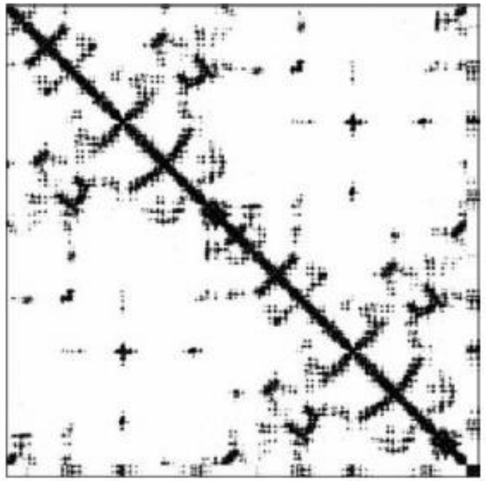
Linear-Scaling DFT and SpGEMM (1)

- Evaluate the **density matrix** P from its functional definition

$$P = \frac{1}{2}\left(I - \mathrm{sign}(S^{-1}H - \mu I)\right)S^{-1}$$

where H is Kohn-Sham matrix, S is the overlap matrix, I is the identity matrix, and μ is the chemical potential

  - The matrices are sparse with a priori unknown sparsity patterns
  - Non-zero elements are small dense blocks, e.g. 13 x 13
  - Typical occupancies >10% (up to nearly dense)
  - On-the-fly filtering procedure during the product of two dense blocks



Linear-Scaling DFT and SpGEMM (2)

- The matrix sign function is defined as

$$\mathrm{sign}(A) = A(A^2)^{-1/2}$$

- Compute with a simple iterative scheme

$$X_0 = A \cdot ||A||^{-1}$$
$$X_{n+1} = \frac{1}{2} X_n(3I - X_n^2)$$
$$X_\infty = \mathrm{sign}(A)$$

→ **Requires SpGEMM** (two multiplications per iteration)

- SpGEMM accounts for >80% of the total runtime of the simulations

The DBCSR library

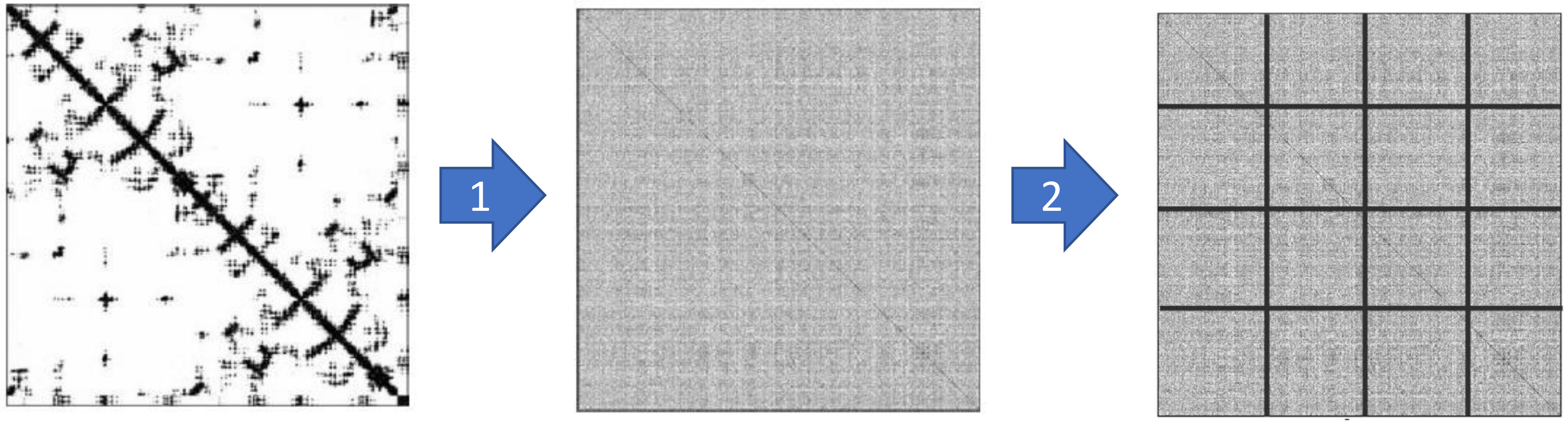
- Standalone library implemented in Fortran (https://dbcsr.cp2k.org)
  - Distributed Block Compressed Sparse Row

Address the following requirements:

1. Take full advantage of the block-structured sparse nature of the matrices, including on-the-fly filtering
2. The dense limit as important as the sparse limit
3. Provide good scalability for a large number of processors

Distribution and Decomposition

1. Independent permutation of row and column block indices to achieve a good load balance
   - Each processor holding approximately the same amount of data, with roughly the same amount of Flops
   - Static decomposition for all multiplications
2. 2D grid decomposition over P processors



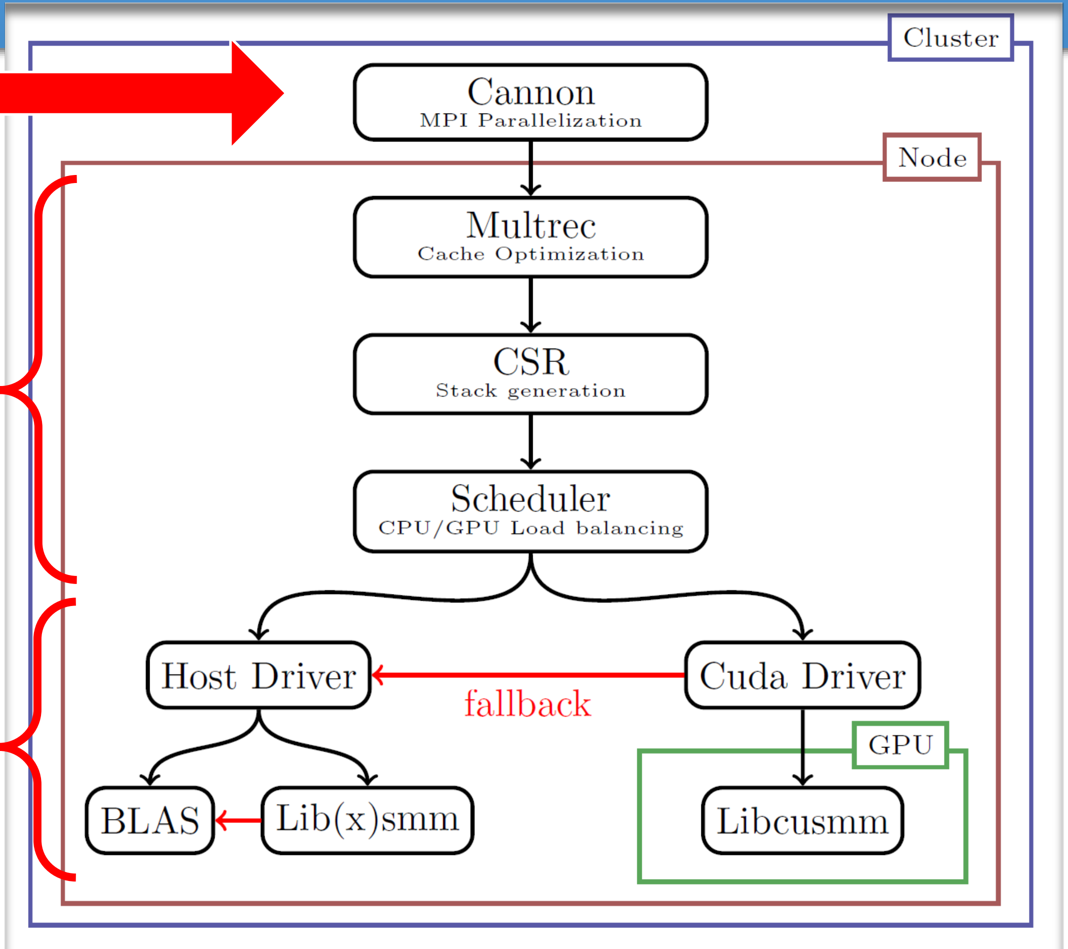
→ Use optimized dense matrix-matrix multiplication algorithm

DBCSR's multiplication scheme

MPI parallelization (focus of this presentation)

Multiplications of blocks are organized in stacks

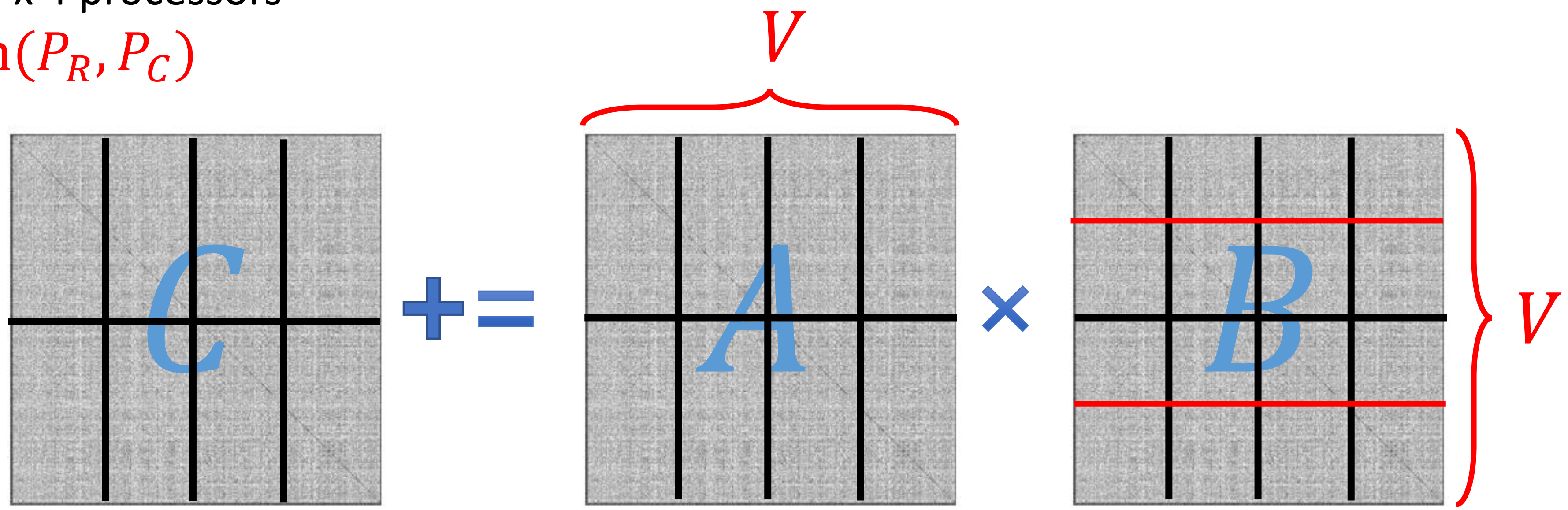
OpenMP parallelization



- Libsmm and Libcusmm are part of the library
- Libxsmm developed by Intel (https://github.com/hfp/libxsmm)

Cannon's Algorithm $C += A\,B$ (1)

- Data is decomposed such that C is always local, i.e. it does not require communications
- Generalize to an arbitrary 2D processor grid $P = P_R \cdot P_C$
  - Introducing a virtual topology
    - E.g. 2 x 4 processors
  - $V = \mathrm{lcm}(P_R, P_C)$



L. E. Cannon. 1969. *A cellular computer to implement the Kalman Filter Algorithm*. Ph.D. Dissertation. Montana State University

Cannon's Algorithm $C += A\ B$ (2)

- V steps for each multiplication (ticks)
  - Minimal when $P_R = P_C$ (square topology) or at least when they have most of their factors in common
    - V scales as $O(\sqrt{P})$
- Per each tick
  1. Data transfer for A and B between grid *processors* neighbors
  2. Local multiplication and accumulation

  ➔ Communication and computation overlap
- The volume of communicated data by each processor scales as $O(1/\sqrt{P})$
- Two buffers per each processor for matrices A and B used for communication and computation

Overview

- Sparse Matrix-Matrix Multiplication
  - Focus on Linear Scaling Density Functional Theory
  - Introducing Distributed Block Compressed Sparse Row (DBCSR) library
- New 2.5D Algorithm
  - Comparison with DBCSR Cannon's algorithm
- One-sided MPI implementation
- Performance
- Conclusion and outlook

2.5D Algorithm

- Decompose data in 2D processor grid $P = P_R \cdot P_C$
  - Same as existing DBCSR algorithm
- Computation runs in a 3*D grid* $\sqrt{\frac{P}{L}} \cdot \sqrt{\frac{P}{L}} \cdot L$
  - $1 \leq L \leq \sqrt[3]{P}$ is the number of layers ($L = 1$ is the 2D algorithm)
  - Each processor evaluates L local C parts
  - Communicate A and B and reuse them for local C evaluations, i.e. less communications

E. Solomonik and J. Demmel. 2011. *Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms*. In European Conference on Parallel Processing. Springer, 90–109.

2.5D Algorithm Example

- Computation on a 2 x 4 processors grid
  - Use virtual topology $A(2 \times 4)$ and $B(4 \times 4)$
  - 4 ticks
- $L = 2$, i.e. 2 x 2 x 2 computational grid

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}$$

	1	2	3	4
1	1. $C_{1,1}$+=$A_{1,1}B_{1,1}$ 2. $C_{1,3}$+=$A_{1,1}B_{1,3}$ 3. $C_{1,1}$+=$A_{1,2}B_{2,1}$ 4. $C_{1,3}$+=$A_{1,2}B_{2,3}$		1. $C_{1,3}$+=$A_{1,3}B_{3,3}$ 2. $C_{1,1}$+=$A_{1,3}B_{3,1}$ 3. $C_{1,3}$+=$A_{1,4}B_{4,3}$ 4. $C_{1,1}$+=$A_{1,4}B_{4,1}$	
2				
	Layer 1		Layer 2	

Reuse A

Exchange C: 1 communication per processor

6 A and B communications per processor (8 in the 2D algorithm)

2.5D Algorithm Requirements

- Virtual topology case
  - $mx = \max(P_R, P_C)$ and $mn = \min(P_R, P_C)$, mx multiple integer of mn and $mx \leq mn^2$
    - ➔ Layers along the largest grid dimension
  - Examples with $L = 2$
    - ☹ 9 x 2
    - ☺ 10 x 5 ➔ 5 x 5 x 2
- Square topology
  - L square number and P_R integer multiple of $\sqrt{L}$
    - ➔ Layers along the both grid dimensions
  - Examples with $L = 4$
    - ☹ 9 x 9
    - ☺ 10 x 10 ➔ 5 x 5 x 4

➔ By construction P/L is a square number

2.5D Algorithm Considerations (1)

- Requires $O(L)$ more memory per processor with respect previous DBCSR algorithm
  - L buffers to store partial C (it was 1)
  - $\max(2, \sqrt{L}) + 2$ buffers to communicate/cache A and B (it was 4)

- The volume of communicated data for A and B scales as $O\left(1/\sqrt{PL}\right)$, i.e. reduced by a factor $\sqrt{L}$

→ Trading memory for communications, i.e. reduce the volume of exchanged data by locally caching matrices

2.5D Algorithm Considerations (2)

- $L - 1$ communications of partial C per each processor to get everything in the right processor
- Total amount of data to communicate (S_X is the size of the matrix X)

$$\underbrace{\frac{V}{\sqrt{L}}(S_A + S_B)}_{A,\,B \text{ panels}} + \underbrace{(L-1)S_C}_{C \text{ panels}}$$

- For the sparse case $S_C > S_{A,B}$, therefore the C data exchange can be dominant for large L

Overview

- Sparse Matrix-Matrix Multiplication
  - Focus on Linear Scaling Density Functional Theory
  - Introducing Distributed Block Compressed Sparse Row (DBCSR) library
- New 2.5D Algorithm
  - Comparison with DBCSR Cannon's algorithm
- One-sided MPI implementation
- Performance
- Conclusion and outlook

One-Sided MPI Implementation

- Initial local data for matrices A and B organized in memory pools
  - Reused between multiplications
  - Reallocation only if the required size is larger than actual size

- Create MPI Windows attached to the memory pools
  - Avoid unnecessary creation/free of the windows when there is no reallocation of the memory pools

Previous VS New DBCSR MPI Implementation

- **Previous:** implementation for the Cannon's algorithm based on Point-to-Point communications (**`mpi_isend/mpi_irecv`**)
- **New:** use RMA passive target to access the data in the initial position (**`mpi_rget`**)
  - Read-only data, no neighbor communications
  - One more buffer per A and B to store the initial data
- New implementation requires less synchronization during the multiplications
  - Just check the communication request on the receiver
- Overall the new implementation is more flexible

Overview

- Sparse Matrix-Matrix Multiplication
  - Focus on Linear Scaling Density Functional Theory
  - Introducing Distributed Block Compressed Sparse Row (DBCSR) library
- New 2.5D Algorithm
  - Comparison with DBCSR Cannon's algorithm
- One-sided MPI implementation
- Performance
- Conclusion and outlook

Benchmarks

- 3 *real* benchmarks taken from the CP2K simulation framework (http://www.cp2k.org)

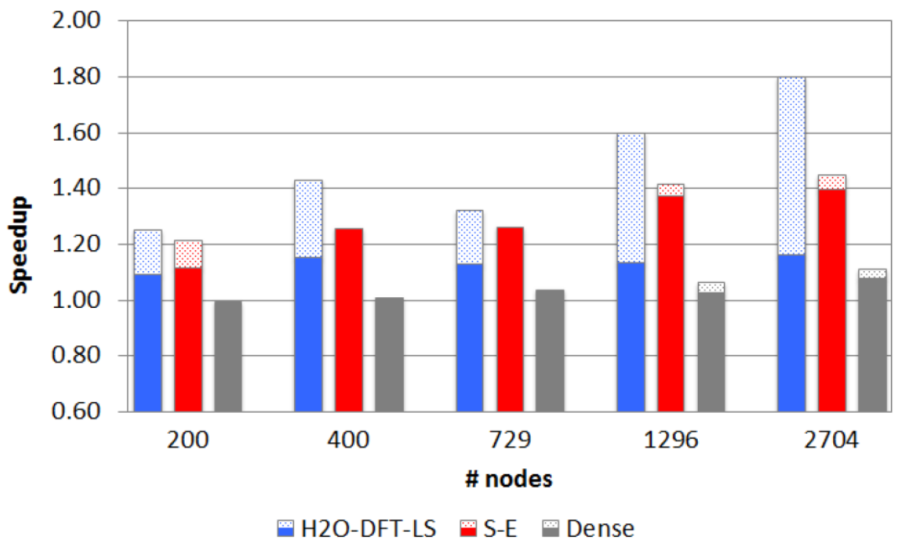
	H2O-DFT-LS	S-E	DENSE
Average Occupancy (%)	10	0.06	100
Block size	23 x 23	6 x 6	32 x 32
# Multiplications	193	1198	10

- Test on Piz Daint @ CSCS (CRAY XC30, Sandy Bridge + NVIDIA K20x)
  - Single rank per node and 8 OpenMP threads + GPU
  - CRAY MPI + DMAPP for fast RMA communicatons

Strong scaling results

- PTP = Point-to-Point, i.e. previous DBCSR implementation
- OS*L* = One-sided with *L* layers, i.e. new DBCSR implementation

	# nodes	H2O-DFT-LS					S-E					Dense				
		PTP	OS1	OS2	OS4	OS9	PTP	OS1	OS2	OS4	OS9	PTP	OS1	OS2	OS4	OS9
DBCSR execution time (seconds)	200	325	298	260	–	–	558	500	459	–	–	42.8	43.0	43.9	–	–
	400	212	184	–	148	–	390	310	–	310	–	22.1	21.9	–	23.6	–
	729	155	137	–	–	117	310	246	–	–	314	13.3	13.3	–	–	15.5
	1296	136	120	–	85	92	282	205	–	199	254	11.2	10.9	–	10.5	11.6
	2704	99	85	–	55	–	249	178	–	172	–	10.8	10.0	–	9.7	–



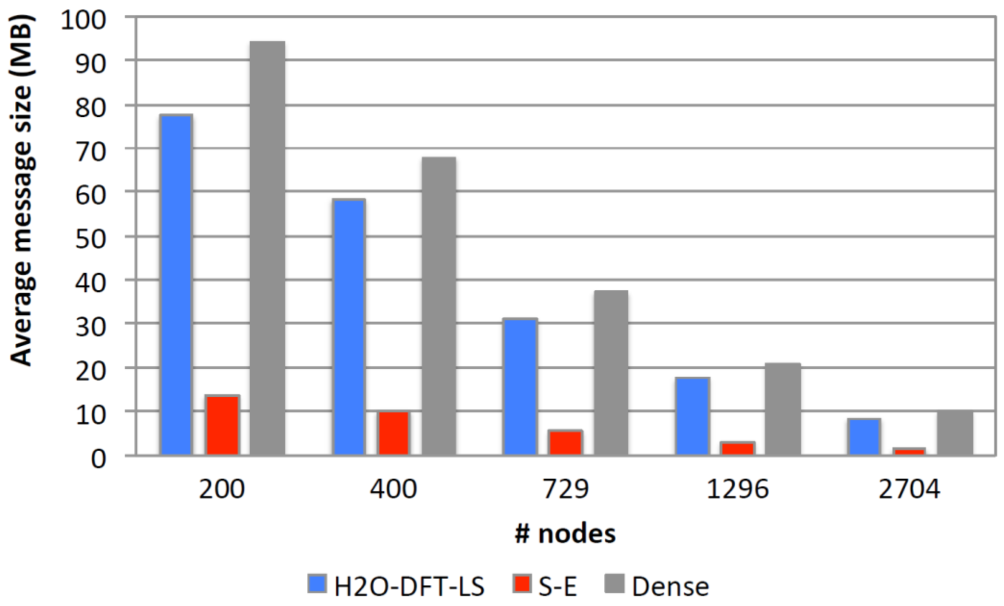
Solid bars: PTP/OS1
Shaded bars: PTP/min(OS*L*)

In average:
- H2O-DFT-LS: OS*L* (*L* > 1) fastest
- S-E: OS1 fastest
- Dense: no significant speedup

As expected speedups improve with higher number of nodes

Strong scaling results considerations

- Correlate speedups with message size of A and B communications and computation (PTP, OS1)



- H2O-DFT-LS:
  - ➔ Communication ☹ **VS** Computation ☺
  - ➔ Communication limited
  - ➔ Improve performance with OSL ($L > 1$)
- S-E:
  - ➔ Communication ☺ **VS** Computation ☺
  - ➔ OS performs very well with small messages
  - ➔ No communication limited already with OS1
- Dense:
  - ➔ Communication ☹ **VS** Computation ☹
  - ➔ No communication limited

- Memory footprint under control (<8GB per processor)

Overview

- Sparse Matrix-Matrix Multiplication
  - Focus on Linear Scaling Density Functional Theory
  - Introducing Distributed Block Compressed Sparse Row (DBCSR) library
- New 2.5D Algorithm
  - Comparison with DBCSR Cannon's algorithm
- One-sided MPI implementation
- Performance
- Conclusion and outlook

Conclusion and Outlook

- Introducing the 2.5D multiplication algorithm in DBCSR for sparse matrix-matrix multiplications improves the performance with respect to previous DBCSR implementation by reducing the volume of exchanged data
  - The speedup becomes larger when we use more processors, up to 1.8x
  - One-sided MPI communications gives better performance and it is also more flexible than Point-to-point communications

- The project will be further extended under a new PASC project (2017-2020)
  - Tensor algebra, more details at the tomorrow POSTER session
  - We are looking for a postdoc, if you are interested (or you know a possible candidate) please talk to me or my colleagues

References

- Urban Borštnik *et al.*, *Sparse matrix multiplication: The distributed block-compressed sparse row library*, Parallel Computing, 2014, Volume 40, Issues 5–6, pp 47–58
- Ole Schütt *et al.*, *GPU Accelerated Sparse Matrix Matrix Multiplication for Linear Scaling Density Functional Theory*, chapter in "Electronic Structure Calculations on Graphics Processing Units", John Wiley and Sons, ISBN 9781118661789
- Proceeding of this conference
- http://dbcsr.cp2k.org
- http://cp2k.org

Thanks!
Questions?

Thanks to CSCS for providing access to Piz Daint, and the PASC project for funding the activity