University of Zurich UZH

DBCSR

Porting the DBCSR library for Sparse Matrix-Matrix Multiplications to Intel Xeon Phi systems

Jürg Hutter, Alfio Lazzaro, Ilia Sivkov

University of Zürich (CH)

IXPUG
THE INTEL® XEON PHI™ USERS GROUP

IXPUG Europe Spring 2018 @ CINECA, Bologna, Italy

Overview

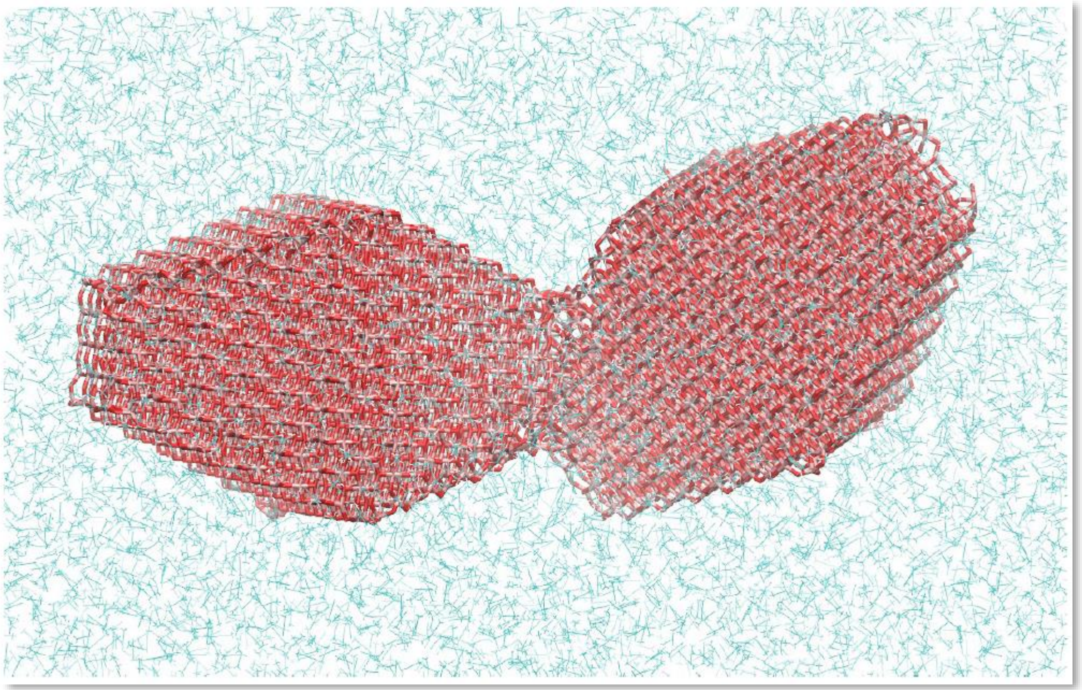
- Sparse Matrix-Matrix Multiplication (SpGEMM)
  - Focus on Linear Scaling Density Functional Theory
- Introducing Distributed Block-Compressed Sparse Row (DBCSR) library
  - OpenMP and MPI parallelization
  - CUDA parallelization
- Performance results on Intel Xeon Phi (KNL)
  - Time-to-Solution (TTS) and Energy-to-Solution (ETS)
- Performance comparison
  - Intel Xeon, Intel Xeon+GPU
- Conclusion and outlook

Overview

- □ Sparse Matrix-Matrix Multiplication (SpGEMM)
  - □ Focus on Linear Scaling Density Functional Theory
- □ Introducing Distributed Block Compressed Sparse Row (DBCSR) library
  - □ OpenMP and MPI parallelization
  - □ CUDA parallelization
- □ Performance results on Intel Xeon Phi (KNL)
  - □ Time-to-Solution (TTS) and Energy-to-Solution (ETS)
- □ Performance comparison
  - □ Intel Xeon, Intel Xeon+GPU
- □ Conclusion and outlook

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)

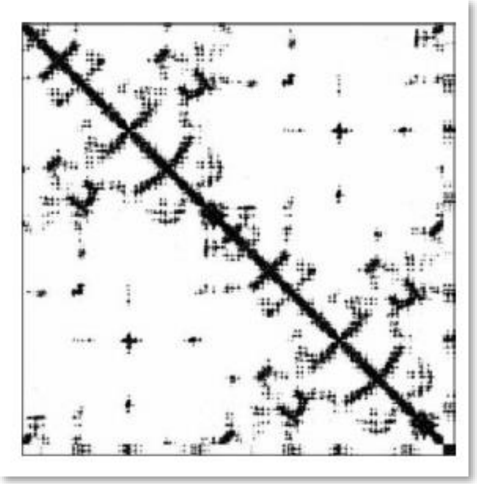
Linear-Scaling DFT and SpGEMM (1)

- 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. 23 x 23
- 👉 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)

  - Sparsity can change between multiplications

- SpGEMM accounts up to 80% of the total runtime of the simulations

Overview

- ❑ Sparse Matrix-Matrix Multiplication (SpGEMM)
  - ❑ Focus on Linear Scaling Density Functional Theory
- ❑ Introducing Distributed Block Compressed Sparse Row (DBCSR) library
  - ❑ OpenMP and MPI parallelization
  - ❑ CUDA parallelization
- ❑ Performance results on Intel Xeon Phi (KNL)
  - ❑ Time-to-Solution (TTS) and Energy-to-Solution (ETS)
- ❑ Performance comparison
  - ❑ Intel Xeon, Intel Xeon+GPU
- ❑ Conclusion and outlook

The DBCSR library

- Standalone library implemented in Fortran 2003 (https://dbcsr.cp2k.org)
  - Distributed Block-Compressed Sparse Row

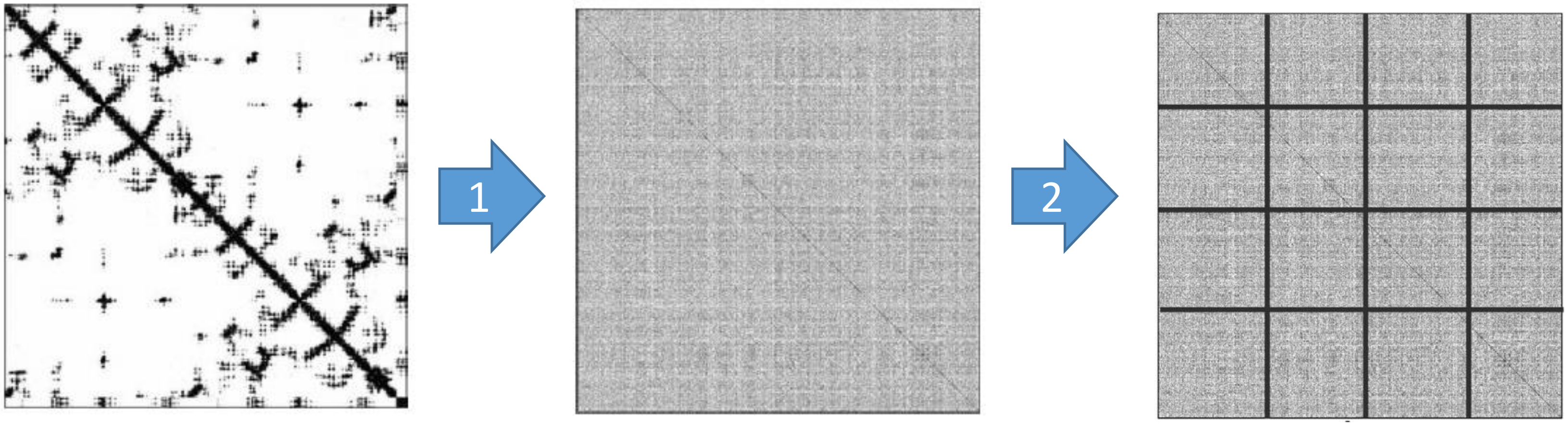
Address the requirements:

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. Random 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

2. 2D grid decomposition over P processes



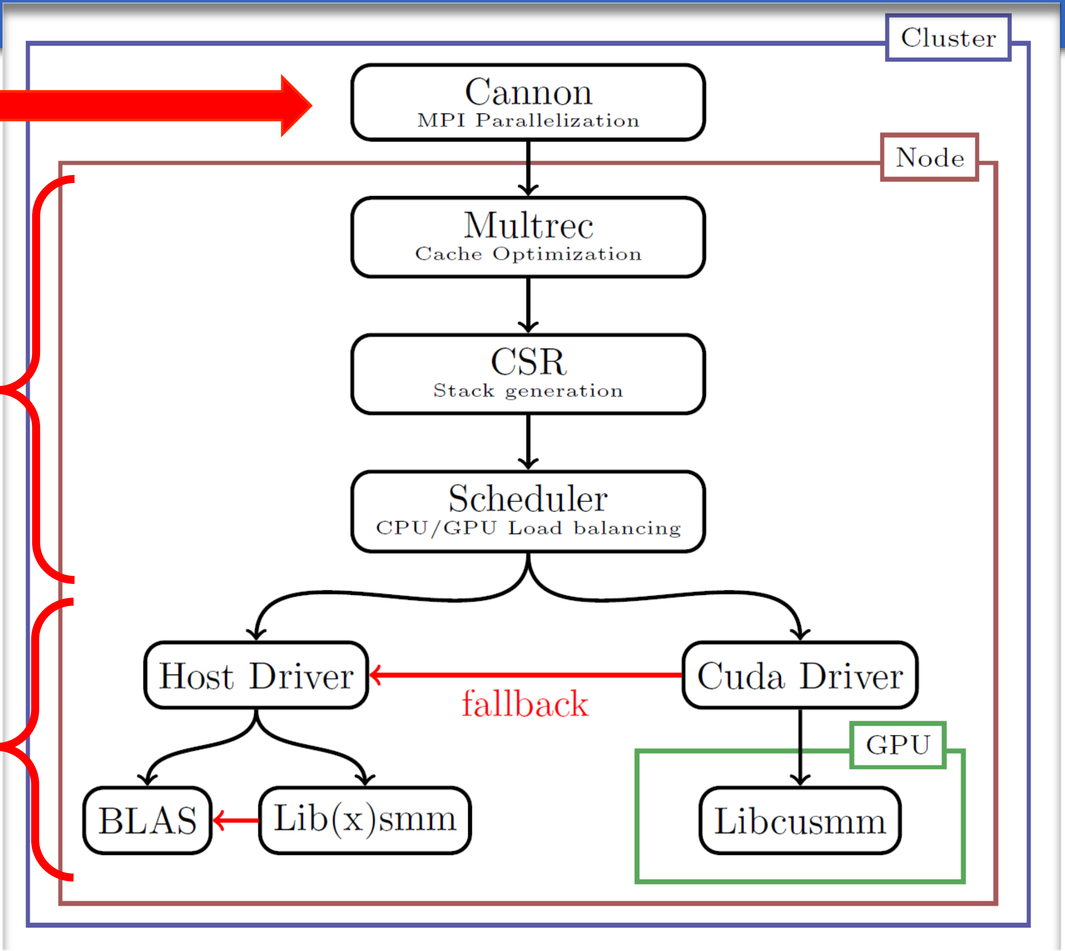
→ Use optimized dense matrix-matrix multiplication algorithm

DBCSR’s multiplication scheme

MPI Parallelization

Multiplications of blocks organized in batches, **partially OpenMP parallelized**

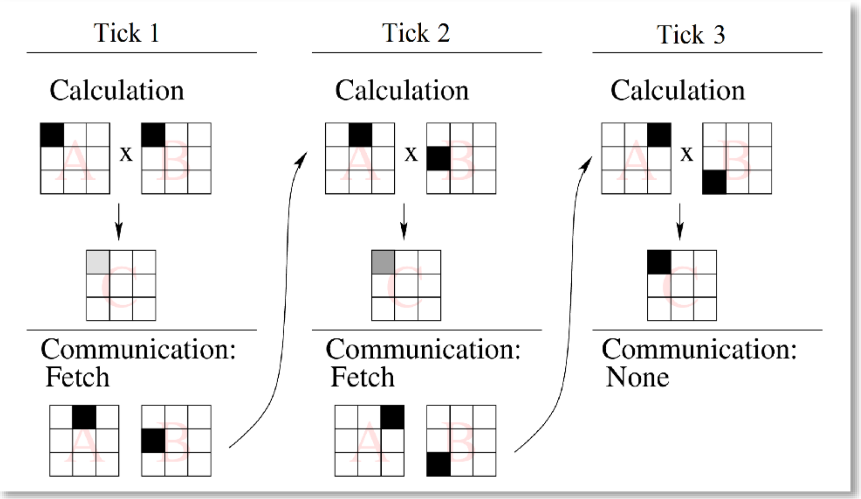
Small matrix multiplications (SMM), **OpenMP parallelized**



- LIBCUSMM is part of the DBCSR library
- LIBXSMM developed by Intel (https://github.com/hfp/libxsmm)

Cannon’s Algorithm $C += A\ B$

- Data is decomposed such that C is always local, i.e. it does not require communications
- $O(\sqrt{P})$ steps (“Ticks”) per each multiplication



```
do i=1,nticks
  call mpi_waitall() - ensures communication
    from previous iteration is complete
    (new data has arrived in current calc
    buffer, comm buffer data has been sent)

  post mpi_irecv() and mpi_isend() for column
    and row shifts - data is sent
    from the current calc buffer,
    and received into the comm buffer

  perform C += A x B on current calc buffers

  comm and calc buffers are (pointer)
  swapped for next iteration

end do
```

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

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

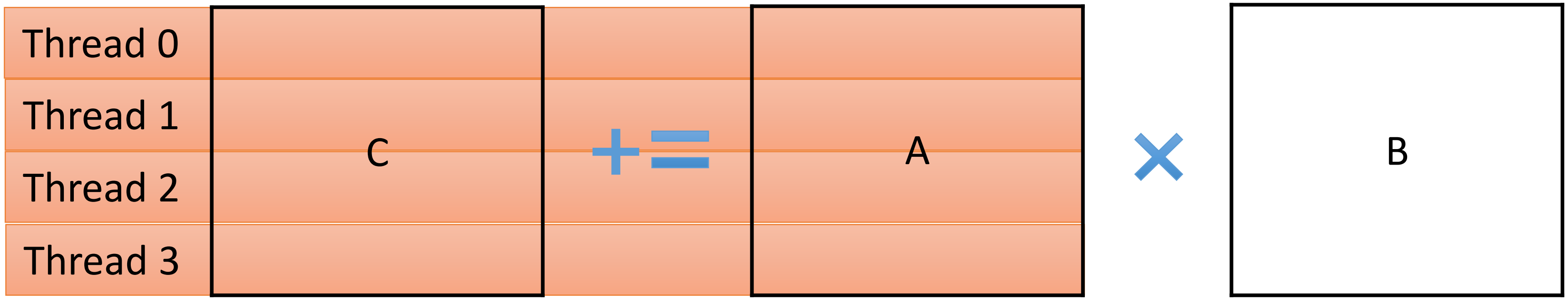
- Data is decomposed such that C is always local, i.e. it does not require communications
- $O(\sqrt{P})$ steps per each multiplication, where per each step:
  1. Data transfer for A and B using non-blocking MPI calls (MPI funneled mode)
  2. Local multiplication and accumulation

  ➔ Communication and computation overlap
- The volume of communicated data by each process scales as $O(1/\sqrt{P})$
  - The communication fraction increases with the number of MPI ranks for a given number of nodes ➔ **keep low the number of ranks/node**

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

OpenMP parallelization

- Local computation consists of the pairwise multiplications of small dense matrix blocks
  - Dimensions: $(m \times k)$ for A blocks, $(k \times n)$ for B blocks
- Corresponding multiplications are organized in batches
  - *Static* assignment of batches with given A matrix row-block indices to OpenMP threads is employed in order to avoid race conditions



OpenMP parallelization

- Local computation consists of the pairwise multiplications of small dense matrix blocks
  - Dimensions: $(m \times k)$ for A blocks, $(k \times n)$ for B blocks
- Corresponding multiplications are organized in batches
  - *Static* assignment of batches with given A matrix row-block indices to OpenMP threads is employed in order to avoid race conditions
  - Cache oblivious matrix traversal to fix the order in which matrix blocks need to be computed
- Batches computed in parallel on the CPU by means of OpenMP threads or alternatively executed on a GPU
  - When the GPU is fully loaded, computation may be simultaneously done on the CPU

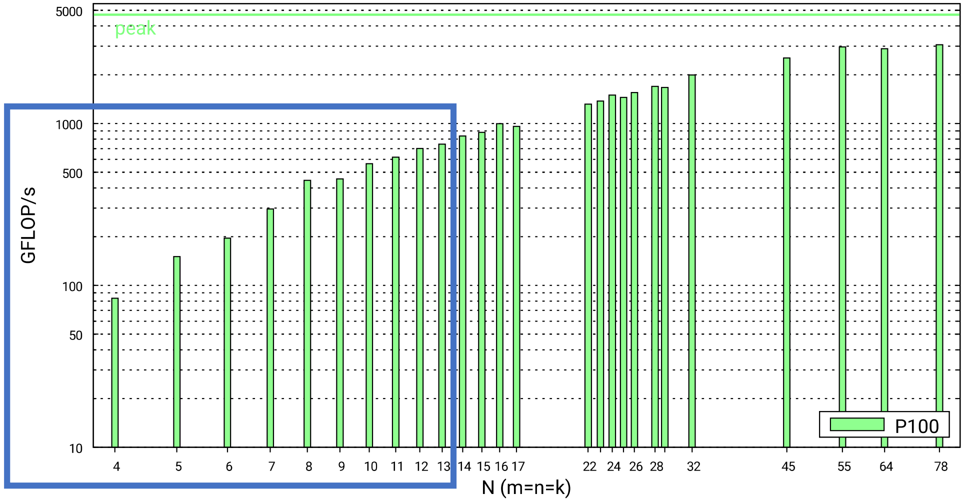
Local small blocks multiplications

- Optimized libraries were developed that outperform vendor BLAS libraries for SMM
  - LIBXSMM for CPU/KNL systems (Intel architectures)
  - LIBCUSMM for Nvidia GPUs with CUDA

- LIBXSMM generates executable code Just-In-Time (JIT) by assembling the instructions in-memory
  - All flavors of AVX extensions are supported
  - Tests with a mini-app, which mimics DBCSR batch multiplications of a series of kernels of interested, show an average speed-up of 2.9x for LIBXSMM over DGEMM-MKL on KNL (peaks at 1.9 TF/s for $m = n = k = 32$ kernel)

CUDA Implementation

- A double-buffering technique, based on CUDA streams and events, is used to maximize the occupancy of the GPU and to hide the data transfer latency
    - Overlap with MPI communications
- LIBCUSMM employs an auto-tuning framework to find optimal kernel for each set of SMM dimensions
    - Speedup in the range of 2–4x with respect to batched DGEMM in cuBLAS
- In absolute numbers, KNL yields higher absolute performance for smaller kernel sizes



Breakdown Execution Summary

A. Time spent in waiting data to arrive (MPI_Waitall for A and B matrices data)
   - Communication time that does not overlap with computation

B. Time spent in the batches execution
   - LIBXSMM/LIBCUSMM executions
   - Compute-intensive, vectorized

C. Time spent in all the rest
   - Initialization/finalization of the multiplications
     - Preparation of the batches
     - Communication from/to GPU
   - Memory-intensive

Overview

- ❑ Sparse Matrix-Matrix Multiplication (SpGEMM)
  - ❑ Focus on Linear Scaling Density Functional Theory
- ❑ Introducing Distributed Block Compressed Sparse Row (DBCSR) library
  - ❑ OpenMP and MPI parallelization
  - ❑ CUDA parallelization
- ❑ Performance results on Intel Xeon Phi (KNL)
  - ❑ Time-to-Solution (TTS) and Energy-to-Solution (ETS)
- ❑ Performance comparison
  - ❑ Intel Xeon, Intel Xeon+GPU
- ❑ Conclusion and outlook

KNL System

- Grand Tavé @ CSCS (CH)
  - 164 Cray XC40 compute nodes, with Intel Xeon Phi 7230 (64 cores @ 1.3 GHz)
  - 96 GB RAM, 16 GB HBM
  - Aries routing and communications ASIC with Dragonfly network topology

Benchmarks

- 3 benchmarks taken from the CP2K simulation framework (http://www.cp2k.org)
  - Representative of large-scale and long-running science runs, hundreds of multiplications

	S-E	H2O-DFT-LS	AMORPH
Average Occupancy (%)	0.06	10	60
Block sizes (m, n, k)	{6}	{23}	{5,13}
# Rows/columns	1,119,744	158,976	141,212

- Only performance of the DBCSR multiplication part
  - ETS based on Cray's power management database
  - We did not perform any lower-level measurements of performance, such as based on hardware event counters
  - Fluctuation up to 5% (averages of 4 independent runs)

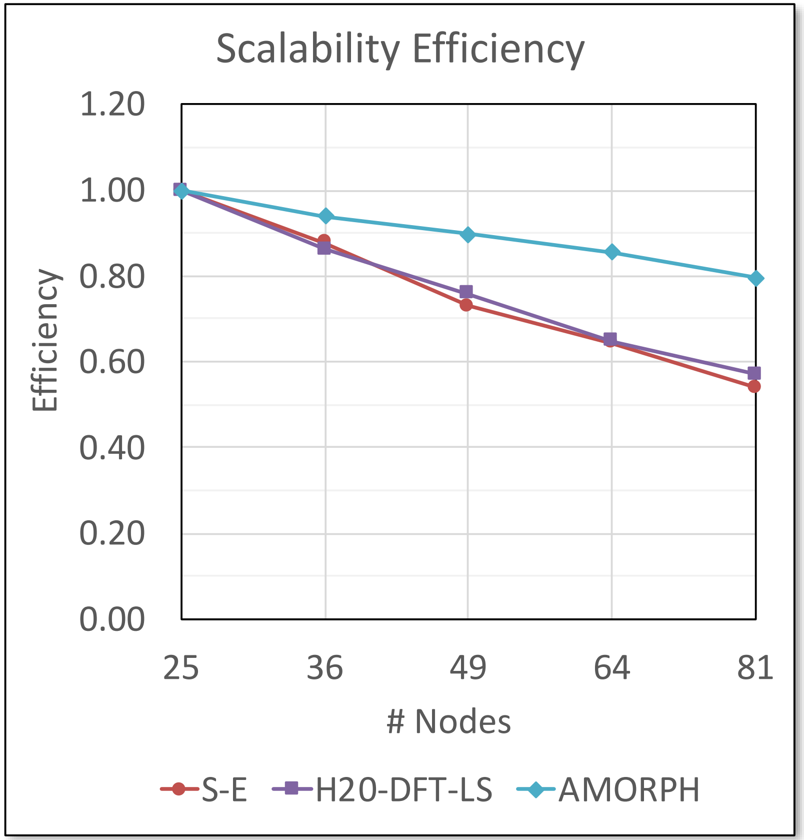
Configuration

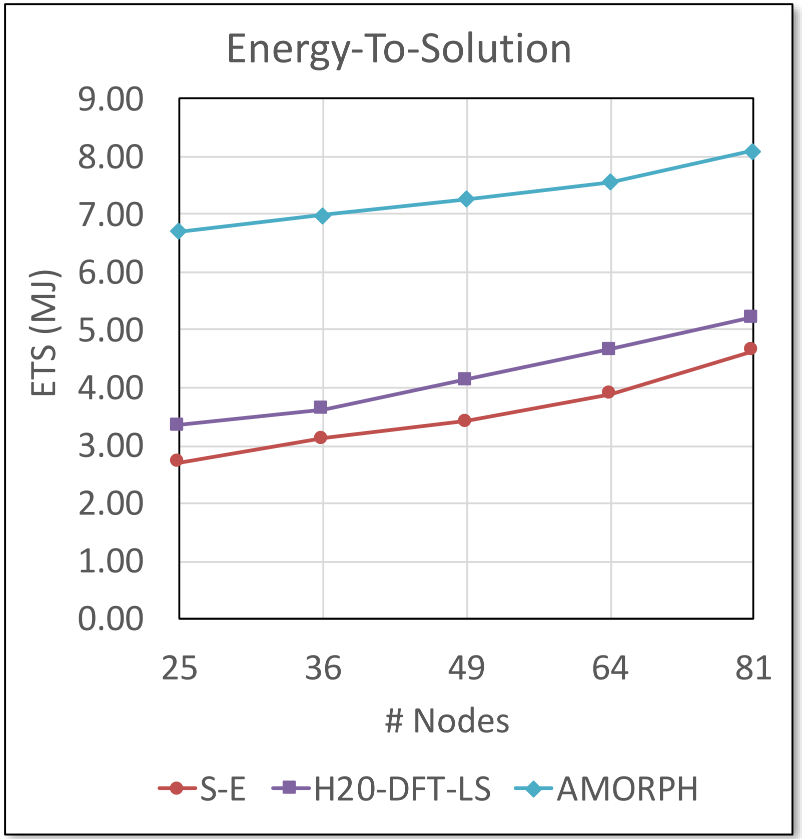
- Code compiled with Intel Fortran Compiler 17.0.4
  - Similar performance with GFortran 7.1.0
- Best performance with 4 MPI ranks and 16 threads per node
  - Multiple threads in core (HT) does not give any speed-up
- All tests are executed in full CACHE mode for the MCDRAM management and QUADRANT clustering mode
- Note that the entire CP2K application requires < 16 GB per node, therefore it fits entirely in MCDRAM
  - No significant speed-up when requiring the application to run in MCDRAM (by using FLAT mode and forcing all allocations in MCDRAM)

Results

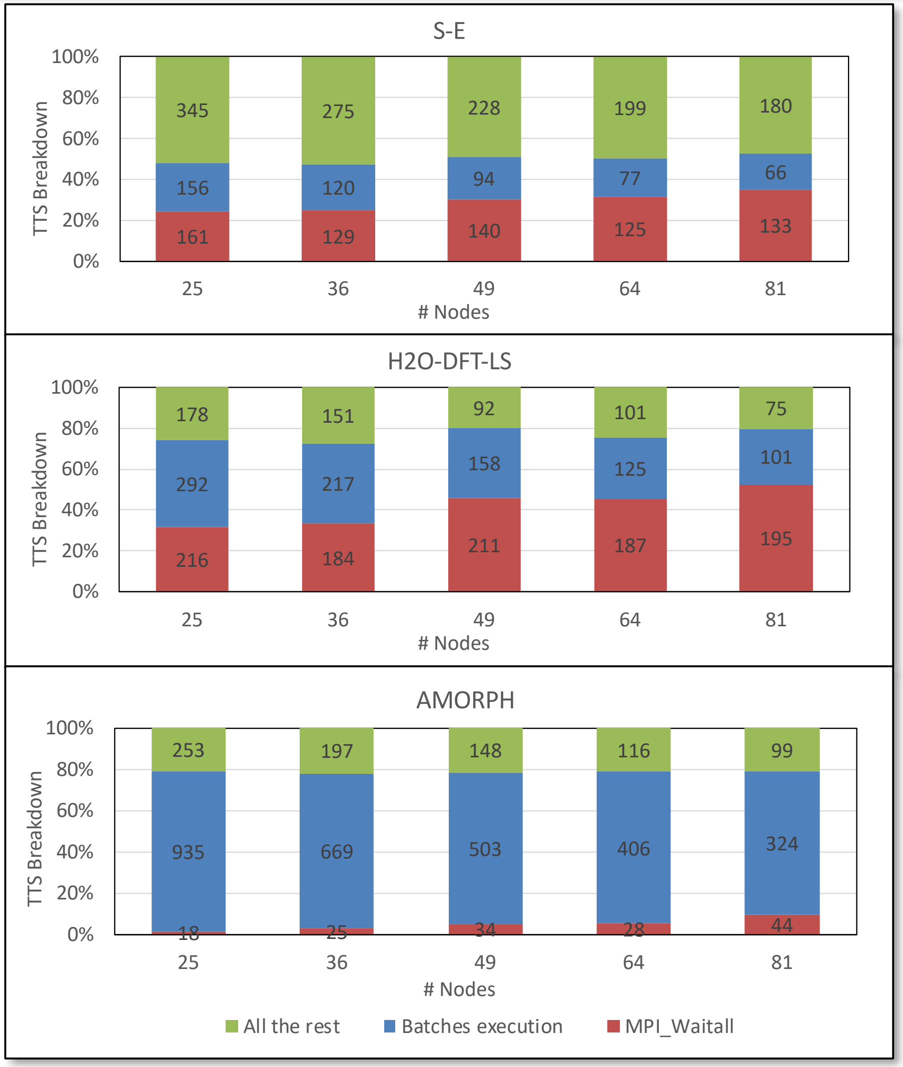
- TTS (s) @ 25 nodes

S-E	H2O-DFT-LS	AMORPH
661	686	1205





TTS Breakdown



Absolute values (seconds) inside each bar part

- S-E
  - Small blocks size
  - Low occupancy
  - Dominated by batches preparation and communications
- H2O-DFT-LS
  - Large blocks size
  - Medium occupancy
  - Communication-bound
- AMORPH
  - Medium blocks sizes
  - High occupancy
  - Computation-bound

Overview

- ❑ Sparse Matrix-Matrix Multiplication (SpGEMM)
  - ❑ Focus on Linear Scaling Density Functional Theory
- ❑ Introducing Distributed Block Compressed Sparse Row (DBCSR) library
  - ❑ OpenMP and MPI parallelization
  - ❑ CUDA parallelization
- ❑ Performance results on Intel Xeon Phi (KNL)
  - ❑ Time-to-Solution (TTS) and Energy-to-Solution (ETS)
- ❑ Performance comparison
  - ❑ Intel Xeon, Intel Xeon+GPU
- ❑ Conclusion and outlook

Systems and Configurations

1. Piz Daint – GPU @ CSCS (CH)

- 5,230 Cray XC50 with Intel Xeon E5-2690 v3 `Haswell` (12 cores, single socket @ 2.6 GHz) and Nvidia Tesla P100
- A single MPI rank and 12 threads per node (no HT)
- GFortran 5.3.0, CUDA 8

2. Piz Daint – MC @ CSCS (CH)

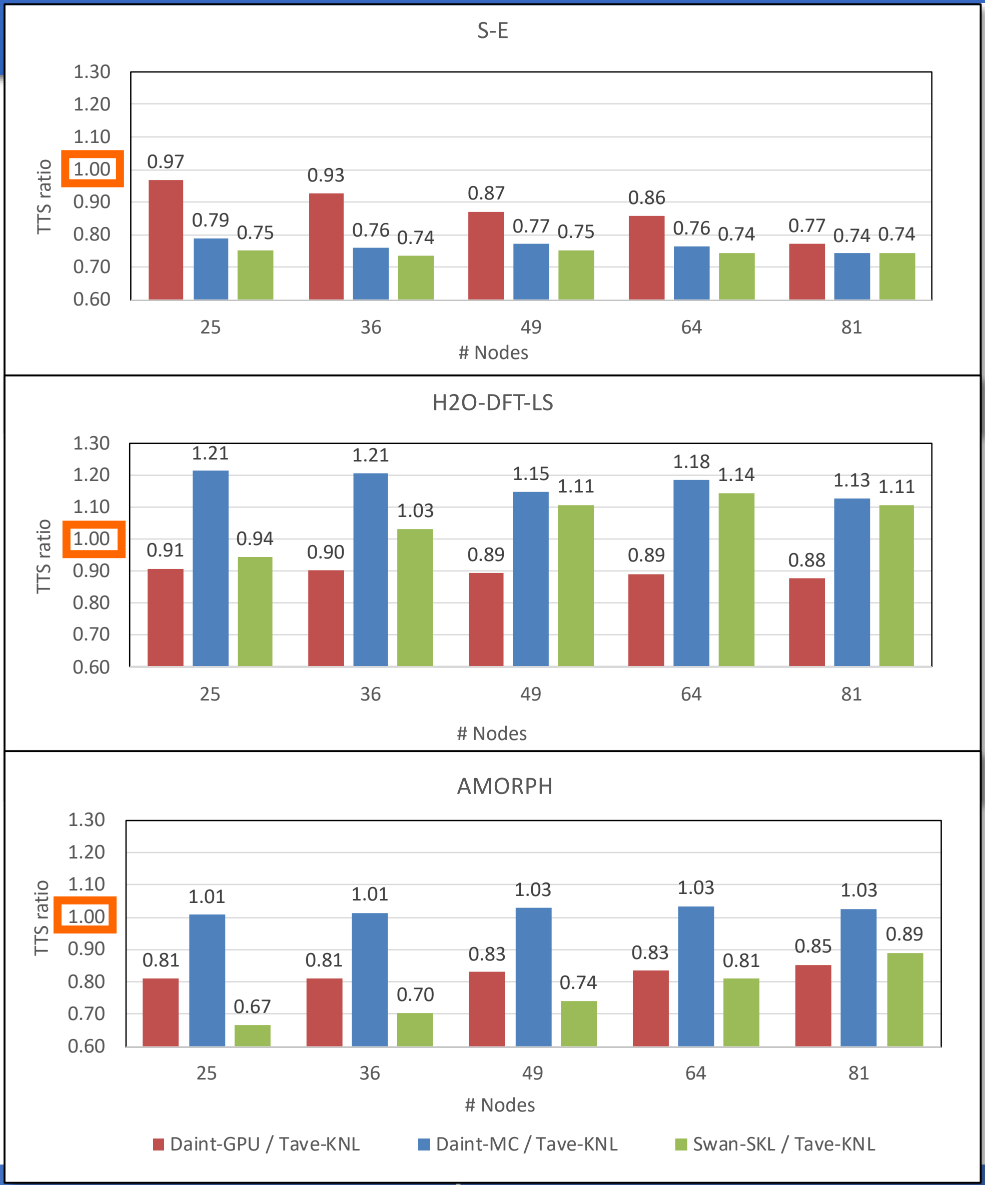
- 1,431 Cray XC40 with Intel Xeon E5-2695 v4 `Broadwell` (18 cores, dual-socket @ 2.1 GHz)
- 4 MPI ranks and 9 threads (no HT)
- GFortran 7.1.0

3. Swan – SKL28 @ Cray

- Cray XC40 with Intel Xeon Platinum 8176 `Skylake` (28 cores, dual-socket @ 2.1 GHz)
- 4 MPI ranks and 14 threads (no HT)
- GFortran 7.3.0

- All systems: Aries network

TTS Comparison



➢ S-E

- Small blocks size
  - Not optimal on GPU
- Dominated by batches preparation and communications
  - Daint-GPU 2x less data to communicate (1 rank/node instead of 4 ranks/node)

➢ H2O-DFT-LS

- Large blocks size
  - Optimal on GPU and KNL
- Communication-bound

➢ AMORPH

- Medium blocks sizes
- Computation-bound

>1 → Tave-KNL faster

<1 → Tave-KNL slower

TTS Breakdown Comparison

>1 ➔ Tave-KNL faster
<1 ➔ Tave-KNL slower

- Average over results for all nodes
  - Batches Execution: computation-bound (GPU execution), well threaded

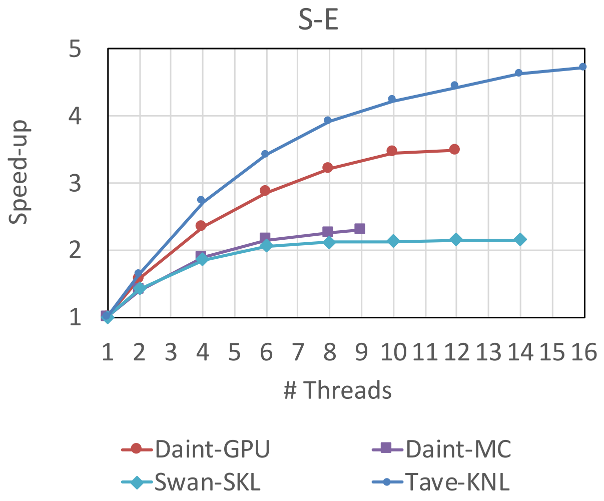
	S-E	H2O-DFT-LS	AMORPH
Daint-GPU / Tave-KNL	0.97	0.62	0.70
Daint-MC / Tave-KNL	0.83	1.62	1.02
Swan-SKL / Tave-KNL	0.47	0.63	0.52

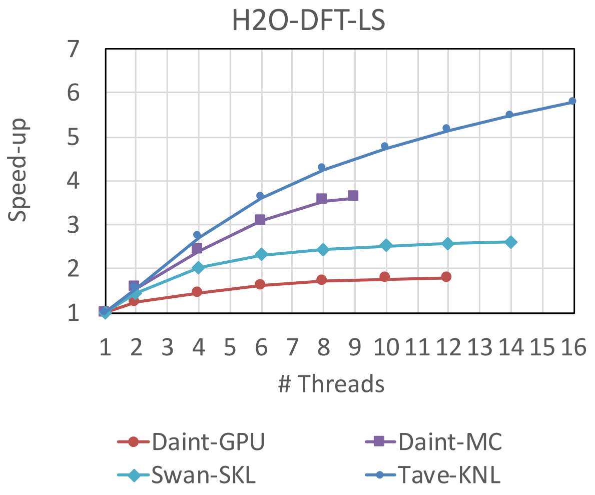
  - All the rest: memory-bound, partially threaded

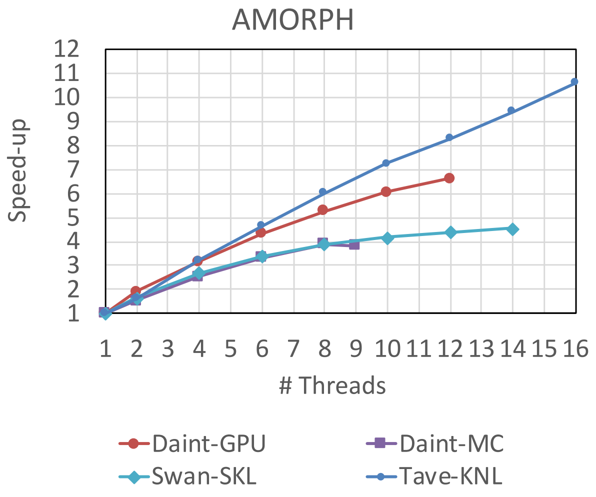
	S-E	H2O-DFT-LS	AMORPH
Daint-GPU / Tave-KNL	0.82	1.27	1.07
Daint-MC / Tave-KNL	0.49	1.13	0.82
Swan-SKL / Tave-KNL	0.44	1.13	0.93

Threading performance

- Speed-up when varying the number of threads with respect to the single thread execution of the DBCSR execution @ 25 nodes
  - The number of MPI ranks is fixed for the corresponding system







- Identified a performance bottleneck (load imbalance) due to the *a priori* static distribution of the SMM among threads

ETS Comparison

- Average over results for all nodes

  >1 → KNL consumes less energy

  <1 → KNL consumes more energy

	S-E	H2O-DFT-LS	AMORPH
Daint-GPU / Tave-KNL	0.78	0.81	0.80
Daint-MC / Tave-KNL	0.95	1.48	1.21
Swan-SKL / Tave-KNL	1.45	1.70	1.21

- Daint-GPU is the most energy-efficient

Overview

- ❑ Sparse Matrix-Matrix Multiplication (SpGEMM)
  - ❑ Focus on Linear Scaling Density Functional Theory
- ❑ Introducing Distributed Block Compressed Sparse Row (DBCSR) library
  - ❑ OpenMP and MPI parallelization
  - ❑ CUDA parallelization
- ❑ Performance results on Intel Xeon Phi (KNL)
  - ❑ Time-to-Solution (TTS) and Energy-to-Solution (ETS)
- ❑ Performance comparison
  - ❑ Intel Xeon, Intel Xeon+GPU
- ❑ Conclusion and outlook

Conclusion and Outlook

- At the same number of nodes, we found that DBCSR executions on a Cray XC40 KNL-based system are:
  - 11%-17% slower and 20% less energy-efficient than on a hybrid Cray XC50 GPU based system with Nvidia P100 cards
  - Up to 17% faster and 70% more energy-efficient that on a Cray XC40 system equipped with dual socket Intel Xeon CPUs
- Bottlenecks (ongoing developments)
  - MPI Communication and load-imbalance
    - Partially implemented a communication optimal algorithm with dynamically distributed load-balancing, implemented with remote memory access MPI communications
  - Threading load-imbalance due to the *a priori* static distribution of the SMM among threads
    - Plan to change the algorithm to be dynamic by using OpenMP tasks (G. Gibb *et al.*, EPCC)

Other 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
- Alfio Lazzaro *et al.*, *Increasing the Efficiency of Sparse Matrix-Matrix Multiplication with a 2.5D Algorithm and One-Sided MPI*. In Proceedings of the Platform for Advanced Scientific Computing Conference, PASC ’17, pages 3:1–3:9, New York, NY, USA, 2017, ACM.
- http://dbcsr.cp2k.org
- http://cp2k.org

Thanks!

Questions?

Thanks to CSCS and Cray for providing access to their systems, the Swiss PASC projects for the funding

Backup

S-E Baselines @ 25 nodes

- TTS Ratio: Row Value / Column Value

	HWS-1T-GPU	HSW-12T	HSW-12T-GPU	BDW-3T	BDW-9T	SKL-3T	SKL-9T	SKL-14T	KNL-3T	KNL-9T	KNL-14T	KNL-16T
HSW-1T	1.27	4.46	4.44	4.03	5.46	4.49	5.70	5.73	2.05	3.69	4.21	4.30
HWS-1T-GPU		3.50	3.49	3.16	4.28	3.52	4.47	4.49	1.61	2.90	3.30	3.37
HSW-12T			1.00	0.90	1.22	1.01	1.28	1.28	0.46	0.83	0.94	0.96
HSW-12T-GPU				0.91	1.23	1.01	1.28	1.29	0.46	0.83	0.95	0.97
BDW-3T					1.35	1.11	1.41	1.42	0.51	0.92	1.04	1.07
BDW-9T						0.82	1.04	1.05	0.38	0.68	0.77	0.79
SKL-3T							1.27	1.28	0.46	0.82	0.94	0.96
SKL-9T								1.00	0.36	0.65	0.74	0.75
SKL-14T									0.36	0.64	0.73	0.75
KNL-3T										1.80	2.05	2.10
KNL-9T											1.14	1.16
KNL-14T												1.02

H2O-DFT-LS Baselines @ 25 nodes

- TTS Ratio: Row Value / Column Value

	HWS-1T-GPU	HSW-12T	HSW-12T-GPU	BDW-3T	BDW-9T	SKL-3T	SKL-9T	SKL-14T	KNL-3T	KNL-9T	KNL-14T	KNL-16T
HSW-1T	8.65	7.33	15.44	6.55	11.58	10.14	14.20	14.85	5.34	10.79	13.30	14.03
HWS-1T-GPU		0.85	1.78	0.76	1.34	1.17	1.64	1.72	0.62	1.25	1.54	1.62
HSW-12T			2.11	0.89	1.58	1.38	1.94	2.03	0.73	1.47	1.81	1.91
HSW-12T-GPU				0.42	0.75	0.66	0.92	0.96	0.35	0.70	0.86	0.91
BDW-3T					1.77	1.55	2.17	2.27	0.82	1.65	2.03	2.14
BDW-9T						0.88	1.23	1.28	0.46	0.93	1.15	1.21
SKL-3T							1.40	1.46	0.53	1.06	1.31	1.38
SKL-9T								1.05	0.38	0.76	0.94	0.99
SKL-14T									0.36	0.73	0.90	0.94
KNL-3T										2.02	2.49	2.63
KNL-9T											1.23	1.30
KNL-14T												1.06

AMORPH Baselines @ 25 nodes

- TTS Ratio: Row Value / Column Value

	HWS-1T-GPU	HSW-12T	HSW-12T-GPU	BDW-3T	BDW-9T	SKL-3T	SKL-9T	SKL-14T	KNL-3T	KNL-9T	KNL-14T	KNL-16T
HSW-1T	2.33	6.85	15.45	6.68	12.45	9.08	16.68	18.84	2.83	7.79	11.11	12.53
HWS-1T-GPU		2.94	6.62	2.86	5.34	3.89	7.15	8.07	1.21	3.34	4.76	5.37
HSW-12T			2.26	0.97	1.82	1.33	2.43	2.75	0.41	1.14	1.62	1.83
HSW-12T-GPU				0.43	0.81	0.59	1.08	1.22	0.18	0.50	0.72	0.81
BDW-3T					1.87	1.36	2.50	2.82	0.42	1.17	1.66	1.88
BDW-9T						0.73	1.34	1.51	0.23	0.63	0.89	1.01
SKL-3T							1.84	2.07	0.31	0.86	1.22	1.38
SKL-9T								1.13	0.17	0.47	0.67	0.75
SKL-14T									0.15	0.41	0.59	0.67
KNL-3T										2.75	3.92	4.42
KNL-9T											1.43	1.61
KNL-14T												1.13