Exascale simulations via the submatrix matrix method


Department of Chemistry
Dynamics of Condensed Matter
Paderborn Center for Parallel Computing
Why Quantum Chemistry?

AB INITIO QUANTUM CHEMISTRY:
A SOURCE OF IDEAS FOR LATTICE GAUGE THEORISTS

Kenneth G. WILSON
The Ohio State University, Department of Physics, 174 W. 18th Avenue, Columbus, OH 43210 USA

Ab initio quantum chemistry is an emerging computational area that is fifty years ahead of lattice gauge theory, a principal competitor for supercomputer time, and a rich source of new ideas and new approaches to the computation of many fermion systems. An overview of the history, current prospects and future frontiers of quantum chemistry is given, with special emphasis on lessons for lattice gauge theory. Particular reference is given to the role of Gaussian basis functions (in place of grids) and analytic (as opposed to Monte Carlo) methods. The main recommendation to lattice gauge theorists is for greater emphasis on infinite momentum frame studies, using Gaussian basis functions.

Mother of HPC Problems

Combine all Levels of the Computer Technology Stack:

- **Algorithm**: Non-orthogonal local submatrix (NOLSM) method compensation of numerical noise
- **Application**: CP2K: quantum chemistry and solid-state theory program
- **Libraries**: NVIDIA cuBLAS with streams and CUDA graphs
- **Computer Architecture**: Third generation NVIDIA tensor cores with mixed FP16/FP32 support
- **Semiconductor Technology**: NVIDIA A100 Tensor-core GPU (TSMC 7 nm FinFET)
Computational Microscope
Schrödinger Equation
„The fundamental laws necessary for the mathematical treatment of large parts of physics and the whole of chemistry are thus fully known ...“

\[ H(r, R)\Psi(r, R) = E\Psi(r, R) \]
Schrödinger Equation

„... the difficulty lies only in the fact that application of these laws leads to equations that are too complex to be solved.“
Schrödinger Equation

... hence it would be desirable to develop practical approximation schemes for the application of quantum mechanics
<table>
<thead>
<tr>
<th>Nuclei</th>
<th>Electrons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Molecular Dynamics (MD)</td>
<td>Ab-Initio MD (AIMD)</td>
</tr>
<tr>
<td>Path-Integral MD (PIMD)</td>
<td>Ab-Initio PIMD (PI-AIMD)</td>
</tr>
</tbody>
</table>

Classical Quantum Mechanical
\[
\mathcal{H}_e(r; R)\psi(r; R) = \varepsilon(R)\psi(r; R)
\]

\[
M_I \dddot{R}_I = -\nabla_{R_I} [\varepsilon(R) + V_{KK}(R)]
\]
CP2K: Overview

- Static Calculations
  Energy & Structure Optimization
  Transition Paths (String, NEB)
  Properties: NMR, EPR & XAS

- Sampling Techniques
  MC, MD & Path-Integral Methods
  RT-TDDFT/Ehrenfest Dynamics
  Accelerated FES: Metadynamics

- Energy & Force Methods
  All-Electron Calculations (GAPW)
  Quickstep: PP Calculations (GPW)
  Post-HF Methods (RPA, MP2, GW)
  DFT/HF Methods (HFX, CDFT)
  Behler-type NN Potentials
  Semiempirical QC & TB Methods
  Classical Molecular Mechanics
  Embedding Methods (IS, QM/MM)

T. D. Kühne et al., JCP 152, 194103 (2020)
CP2K: Overview

• Static Calculations
  Energy & Structure Optimization
  Transition Paths (String, NEB)
  Properties: NMR, EPR & XAS

• Sampling Techniques
  MC, MD & Path-Integral Methods
  RT-TDDFT/Ehrenfest Dynamics
  Accelerated FES: Metadynamics

• Energy & Force Methods
  All-Electron Calculations (GAPW)
  Quickstep: PP Calculations (GPW)
  Post-HF Methods (RPA, MP2, GW)
  DFT/HF Methods (HFX, CDFT)
  Behler-type NN Potentials
  Semiempirical QC & TB Methods
  Classical Molecular Mechanics
  Embedding Methods (IS, QM/MM)

http://www.cp2k.org
CP2K: Recent Development

- Large Scale Computational Kernels
  - PW-based DFT (SIRIUS)
  - Hybrid DFT, MP2 & RPA
  - Linear Scaling Algorithms
- Approximate Molecular Dynamics
  - 2nd Generation Car-Parrinello MD
- Approximate Computing
- Massive Parallelism
  - Mixed MPI / OpenMP
  - GPU / FPGA support
- Sparse Matrix Algebra
  - Distributed Block CSR
  - Cannon‘s Algorithm & SMM
- Open Source
  - 1.5 mio. lines of code
  - > 6500 regression tests

http://www.cp2k.org
Schrödinger Equation

\[ H[C]C = \varepsilon SC \)

\[ \mathcal{O}(N^3) \]

\[ E = \sum_{i=1}^{N_e} \varepsilon_i - V_{dc} = Tr \left[ C^T H C \right] - V_{dc} \]

\[ = \min_P Tr \left[ PH \right]_{PSP=P} - V_{dc} \]

Linear Scaling Self Consistent Field

Guess initial density $\rho$

Calculate matrix $H$ from $\rho$
Costs: $O(N)$, but dominates for small systems

Calculate eigenvectors $\psi_i$ of $H$
Costs: $O(N^3)$

Calculate new density $\rho = \sum_i |\psi_i|^2$

Calculate $\rho$ directly as matrix function of $H$
Costs: $O(N)$

Calculate energy from $\rho$

Dense linear algebra
Sparse linear algebra

LS-SCF entirely based on sparse linear algebra.
The Sign Method

\[ P = \frac{1}{2} \left( I - \text{sign} \left( S^{-1}H - \mu I \right) \right) S^{-1} \]

- matrix sign function and inversion can be evaluated iteratively
  \[
  \text{sign}(x) = \frac{x}{|x|} = \frac{x}{\sqrt{x^2}}
  \]
  \[
  \text{sign}(A) = A \cdot (A^2)^{-1/2}
  \]
  \[
  X_0 = A
  \]
  \[
  X_{i+1} = \frac{1}{2} X_i \cdot (3I - X_i^2)
  \]
  \[
  \text{sign}(A) = \lim_{i \to \infty} X_i
  \]

- linear-scaling approach \( O(N) \)
- only \textbf{multiplications} of distributed sparse matrices are required
- but usually bound by inter-node communication!

The Sign Method

$O(N)$ methods are inevitable for large systems

The DBCSR Library

• Distributed
  MPI parallelization based on Cannon, 2.5D, Carma, or Cosma algorithm
  On node parallelization via OpenMP
• Block Compressed Sparse Row
  Block-sparse, where block corresponds to atoms
• Small matrix-matrix multiplication library on multicore CPUs & GPUs
  libxsmm, libcusmm, libsmm_acc

The DBCSR Library

http://dbcsr.cp2k.org

The DBCSR Library

1. Random permutation of row and column block indices to balance load
   Each processor is approximately holding the same amount of data, with roughly the same amount of Flops
2. 2D grid decomposition over $P$ processors
   Block-sparse, where block corresponds to atoms

The DBCSR Library

- DBCSR is based on blocked structure
  Non-zero elements are small dense blocks, typically 5x5, 13x13, 23x23, ...
  Take full advantage of the block structured sparse nature of the matrices
  Each block corresponds to the interaction between two atoms
- Dense limit is as important as the sparse limit
- Provide good scalability for a large number of processors
SMM Libraries

- Optimized libraries were developed that outperform vendor BLAS for SMM
  LIBXSMM for Intel-based CPU/KNL systems
  LIBCUSMM for Nvidia GPUs using CUDA
  LIBSMM_ACC for Nvidia/AMD GPUs using CUDA and HIP
- LIBXSMM generates executable code just-in-time (JIT) by assembling the
  instructions in-memory
  All flavors of AVX extensions are supported
  Avg. speed-up of approx. 3 for LIBXSMM over MKL-DGEMM on KNL
- LIBCUSMM employs a double-buffering technique, based on CUDA
  streams, to maximize occupancy of the GPU and hide data transfer latency
- LIBSMM_ACC GPU kernels are JIT compiled at runtime
- LICUSMM & LIBSMM_ACC includes an auto-tuning framework to find
  the best parameters for every (m,n,k)-kernel out of > 100k combinations

SMM Libraries

Additional factor 5 speed-up, when using LIBSMM_ACC instead of LIBXSMM on CRAY XC50 node with 12 core Intel Haswell CPU and Nvidia V100 GPU

FPGA-based Noctua@PC2
\[ B_{k+1} = \frac{1}{p} \left[ (p-1)B_k - ((I-B^p_k A)^q - I) B^{1-p}_k A^{-1} \right] \]
\[ p=1 = [I - (I - B_k A)^q] A^{-1}. \]


Approximate Computing

\[ M_I \ddot{R}_I = \underbrace{\mathbf{F}^{BO}_I + \Xi^N_I} - \gamma_N M_I \dot{R}_I \]
\[ = \mathbf{F}^{FPGA}_I - \gamma_N M_I \dot{R}_I \]

\[ \langle \Xi_D I(0) \Xi_D I(t) \rangle = 2\gamma_D M_I k_B T \delta(t) \]

\[ \langle \frac{1}{2} M_I \dot{R}_I^2 \rangle = \frac{3}{2} k_B T \]

Figure 2. Partial pair correlation function for liquid Si at 3000 K with noisy forces introduced by floating-point errors of magnitude $10^{-2}$ (blue), $10^{-1}$ (green), and $10^{0}$ (red). For comparison, the results, as obtained by our reference calculations without noise, are shown in black.

Figure 3. Kinetic energy distribution of liquid Si at 3000 K, as obtained by our simulations using noisy forces (circles). For comparison, the analytic Maxwell distribution is also shown (line).

To further assess the accuracy of the present method, we expanded the autocorrelation of the noisy forces, i.e.,

$$D_{F N I}(0) F_{NI}(t) E(10a) = D_{F I}(0) + \sum_{N I}(0) \alpha F_{NI}(t) + \sum_{N I}(t) E(10b)$$

$$= h F_{I}(0) F_{I}(t) + D_{F I}(0) X_{NI}(t) E(10c) + D_{F I}(t) X_{NI}(0) E + D X_{NI}(0) X_{NI}(t) E.$$

Since the cross-correlation terms between the exact force and the additive white noise were vanishing due to Equation (4), comparing the autocorrelation of the noisy forces $h F_{NI}(0) F_{NI}(t)$ with
the autocorrelation of the exact forces $h_{FI}(0)$ permitted assessing the localization of the last term of Equation (10c). The fact that $h_{FN}(0)F_I(t)$ was essentially identical to $h_{FI}(0)F_I(t)$, as can be seen in Figure 4, implied that $h_{NI}(0)X_I(t)$ was very close to a $\delta$-function as required by the fluctuation-dissipation theorem in order to ensure an accurate canonical sampling of the Boltzmann distribution. In other words, from this, it followed that our initial assumption underlying Equation (7), to model the noise due to a low precision calculation as an additive white noise channel, was justified.

6. Conclusions

We conclude by noting that the presented method was recently implemented in the universal force engine i-PI \[68\], which can be generally applied to all sorts of forces affected by stochastic noise such as those computed by GPUs or other hardware accelerators \[15\]–\[21\], and potentially even quantum computing devices \[69\]–\[72\]. The possibility to apply similar ideas to N-body simulations \[73, 74\] and to combine them with further algorithmic approximations \[75\] is to be underlined and will be presented elsewhere.

Author Contributions: V.R. wrote the code and conducted the calculations, V.R. and M.L. analyzed the data, M.L., C.P. and T.D.K. interpreted the results, M.L., V.R. and T.D.K. wrote the paper, C.P. and T.D.K. conceived the study and directed the project. All authors have read and agreed to the published version of the manuscript.

Funding: The authors would like to thank the Paderborn Center for Parallel Computing (PC\textsuperscript{2}) for computing time on OC\textsuperscript{ULUS} and FPGA-based supercomputer NOCTUA. Funding from Paderborn University’s research award for “Green IT” is kindly acknowledged. This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (Grant Agreement No. 716142) and from the German Research Foundation (DFG) under the project PerficienCC (grant agreement No PL 595/2-1).

Conflicts of Interest: The authors declare no conflict of interest.

References

Submatrix Method

Purpose: Estimate matrix function (e.g. sign or inversion) of a large sparse matrix

Submatrix Method

Step 1: Identify nonzero values in every column

\[ A \]

Submatrix Method

Step 2: Build submatrix $\mathcal{T}_i(A)$ for every column $i$ with only the rows that have non-zero elements

Submatrix Method

Step 3: Apply matrix function $f$ to submatrices $T_i(A)$

Submatrix Method

Step 4: copy resulting columns to result matrix

Submatrix Method

Properties of the Submatrix method:

- large distributed sparse matrix ⇒ many small dense matrices
- suitable for dense linear algebra
- massively parallel
- linear-scaling approach

Non-Orthogonal Local SM

Implementation with GPUs:

\[ S = \]

\[ H_0 = \]

\[ \Rightarrow \text{minimal communication between nodes and CPU-to-GPU} \]

Non-Orthogonal Local SM

Customized "batching" for matrix iterations:

Kernel from cuBLAS

+CUDA streams: to load the full GPU

+CUDA graphs: reduce kernel launch latency

⇒ for intermediate sized matrices (≥ 1000) about 60%-85% of peak can be reached

Pre-Exascale Simulation

- 936 GPU-nodes with each:
  - CPU: 2xAMD EPYC 7402
  - Memory: 512 GB DDR4-3200 RAM
  - GPU: 4 x NVIDIA A100, 40 GB, NVLink3
  - Network: 4 x Mellanox HDR200 InfiniBand ConnectX 6 (200 Gbit/s each)

- Peak TC Performance:
  - FP64: 73 PFLOP/s
  - FP16/FP32: 1170 PFLOP/s

<table>
<thead>
<tr>
<th>Code, Year</th>
<th>Method</th>
<th>Basis</th>
<th>System</th>
<th># Atoms</th>
<th># Cores</th>
<th>Machine</th>
<th>Peak Performance</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPMD [22] 2005</td>
<td>DFT</td>
<td>PW</td>
<td>bulk SiC</td>
<td>1k</td>
<td>1.2k CPU</td>
<td>IBM p690</td>
<td>1.087 TFlop/s</td>
<td>~ 20%</td>
</tr>
<tr>
<td>Qbox [23] 2006</td>
<td>DFT</td>
<td>PW</td>
<td>bulk Mo</td>
<td>8*1k</td>
<td>128k CPU</td>
<td>IBM BlueGene/L</td>
<td>207.3 TFlop/s</td>
<td>56.5%</td>
</tr>
<tr>
<td>LS3DF [24] 2009</td>
<td>DFT</td>
<td>PW</td>
<td>bulk ZnTeO</td>
<td>36k</td>
<td>147k CPU</td>
<td>Cray Jaguar</td>
<td>442 TFlop/s</td>
<td>~ 33%</td>
</tr>
<tr>
<td>CP2K [25] 2012</td>
<td>LS-DFT</td>
<td>GPW</td>
<td>bulk H₂</td>
<td>1m</td>
<td>47k CPU</td>
<td>Cray XT5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONETEP [26] 2014</td>
<td>LS-DFT</td>
<td>NGWF</td>
<td>amyloid fibril trimer</td>
<td>42k</td>
<td>115k CPU</td>
<td>IBM BlueGene/Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSDFT [27] 2014</td>
<td>DFT</td>
<td>RS-FD</td>
<td>Si nanowire</td>
<td>107k</td>
<td>664k CPU</td>
<td>K-Computer</td>
<td>5.48 PFlop/s</td>
<td>51.67%</td>
</tr>
<tr>
<td>LDC-DFT [28] 2014</td>
<td>SS-DFT</td>
<td>RMG-PW</td>
<td>bulk SiC</td>
<td>6.3m</td>
<td>786k CPU</td>
<td>IBM Blue Gene/Q</td>
<td>5.08 PFlop/s</td>
<td>50.5%</td>
</tr>
<tr>
<td>OpenAtom [29] 2016</td>
<td>DFT</td>
<td>PW</td>
<td>periodic MOF</td>
<td>32*424</td>
<td>262k CPU</td>
<td>IBM BlueGene/Q</td>
<td></td>
<td>~ 52%</td>
</tr>
<tr>
<td>MGmol [30] 2016</td>
<td>LS-DFT</td>
<td>FD</td>
<td>bulk H₂O</td>
<td>1.2m</td>
<td>1.6m CPU</td>
<td>IBM BlueGene/Q</td>
<td></td>
<td>~ 39%</td>
</tr>
<tr>
<td>DFT-FE [31] 2019</td>
<td>DFT</td>
<td>FEM</td>
<td>Mg cluster</td>
<td>10.5k</td>
<td>159k CPU</td>
<td>IBM Summit</td>
<td>46 PFlop/s</td>
<td>27.8%</td>
</tr>
<tr>
<td>CONQUEST [32] 2020</td>
<td>LS-DFT</td>
<td>PAO</td>
<td>bulk Si</td>
<td>1m</td>
<td>200k CPU</td>
<td>K-Computer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pre-Exascale Simulation

Strong scaling of bulk water with 102 million atoms:

Pre-Exascale Simulation

Strong scaling of bulk water with 102 million atoms:

- Peak multiply performance (312 TFLOP/s/GPU)
- Performance of NOLSM for bulk water

![Graph showing strong scaling performance](image)

⇒ ≈ 43% of peak reached

⇒ small matrix sizes limit achievable performance

Combination of Submatrices

Idea: use flexibility of submatrix method and combine submatrices with similar columns

\[ \Rightarrow \text{fewer but larger submatrices!} \]

Combination of Submatrices

average submatrix dimension $654 \rightarrow 1202$

and reduction of floating-point operations by factor 1.65!
Combination of Submatrices

Strong scaling of HIV-1 with 62.5 million atoms:

\[ \Rightarrow \approx 68\% \text{ of peak reached} \]

\[ \Rightarrow 324 \text{ PFlops FP16/FP32 for 384 nodes (68\% of TC Peak)} \]

Summary & Outlook

- extended electronic-structure based molecular dynamics simulations to more than 100 million atoms
  - 102 million atoms for bulk water
  - 62 million atoms for HIV-1 capsid
- Non-orthogonal local submatrix (NOLSM) method:
  - for matrix functions of **large sparse matrices**
  - massively parallel **communication-avoiding** method
  - GPU-accelerated for NVIDIA GPUs
    - minimal transfer between host and GPUs
    - matrix construction on GPUs
    - mixed-precision on **NVIDIA tensor cores**
  - compensation schema for numerical noise
  - candidate for one of the first scientific FP16/FP32-exaflop simulations

HIV-1 capsid in aqueous solution with 62 million atoms

Acknowledgements

• Robert Schade, PC2: NOLSM
• Michael Lass, PC2: Submatrix Method
• Tobias Kenter, PC2: Combination of Submatrices
• Christian Plessl, PC2: AC, FPGA
• Ole Schütt, Google: LIBCUSMM
• Valery Weber, IBM ZRL: DBCSR
• Alfio Lazzaro, Cray EMEA Research Lab: DBCSR
• Hans Pabst, Intel Extreme Computing: LIBXSMM
• Joost VandeVondele, CSCS: COSMA, LIBSMM_ACC
• Stephan Mohr, BSC: Submatrix Method
• Matthias Krack, PSI: CP2K
• Jürg Hutter, UZH: CP2K