Scalable solvers for meshless methods on many-core clusters

Peter Zaspel

University of Basel

AGUQ Workshop on Uncertainty Quantification 2018
TU Dortmund, Dortmund, March 12-14, 2018

Objective

Solution of dense system of linear equations

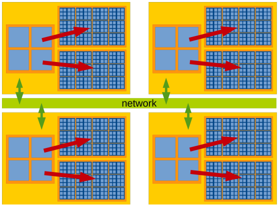
$$\begin{pmatrix} k(\boldsymbol{y}_1, \boldsymbol{y}_1) & \cdots & k(\boldsymbol{y}_1, \boldsymbol{y}_{N_\Gamma}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{y}_{N_\Gamma}, \boldsymbol{y}_1) & \cdots & k(\boldsymbol{y}_{N_\Gamma}, \boldsymbol{y}_{N_\Gamma}) \end{pmatrix} \boldsymbol{x} = \boldsymbol{b}$$

- $X_{N_\Gamma} := \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{N_\Gamma}\} \subset \Gamma \subset \mathbb{R}^d$ set of meshfree points
- $k : \Gamma \times \Gamma \to \mathbb{R}$ positive definite kernel function
- N_Γ potentially **extremely** large

Applications

- UQ
- quadrature
- machine learning

Solvers on many-core clusters



Outline

Motivation

Hierarchical matrices

Many-core parallelization of $\mathcal{H}$ matrices

Solvers on many-core clusters

Outline

Motivation

Hierarchical matrices

Many-core parallelization of $\mathcal{H}$ matrices

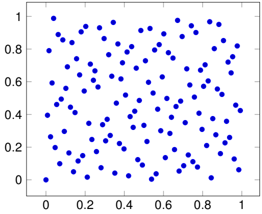
Solvers on many-core clusters

Kernel-based stochastic collocation for CFD [Griebel,Rieger 2015] [Z. 2015]

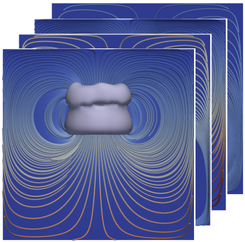
Example: Expectation value

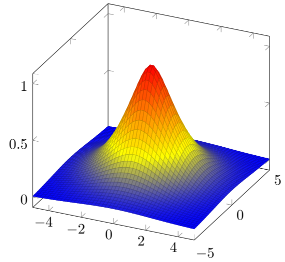
$$\mathbb{E}[u](\boldsymbol{x}) \approx \sum_{s=1}^{N_\Gamma} u(\boldsymbol{y}_s, \boldsymbol{x})\mathbb{E}[L_s] \approx \ldots = \sum_{s=1}^{N_\Gamma} u(\boldsymbol{y}_s, \boldsymbol{x}) \left((A_{k,X_\Gamma})^{-1} \boldsymbol{e} \right)_s$$

$$A_{k,X_\Gamma} = \begin{pmatrix} k(\boldsymbol{y}_1, \boldsymbol{y}_1) & \cdots & k(\boldsymbol{y}_1, \boldsymbol{y}_{N_\Gamma}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{y}_{N_\Gamma}, \boldsymbol{y}_1) & \cdots & k(\boldsymbol{y}_{N_\Gamma}, \boldsymbol{y}_{N_\Gamma}) \end{pmatrix}, \quad \boldsymbol{e} = \begin{pmatrix} \mathbb{E}[k(\cdot, \boldsymbol{y}_1)] \\ \vdots \\ \mathbb{E}[k(\cdot, \boldsymbol{y}_{N_\Gamma})] \end{pmatrix}$$



collocation points $\boldsymbol{y}_s$


solution snapshots $\boldsymbol{u}(\boldsymbol{y}_s, \boldsymbol{x}, t)$



RBF kernel function
$k(\boldsymbol{y}_i, \boldsymbol{y}_j) := \varphi(\|\boldsymbol{y}_i - \boldsymbol{y}_j\|)$

joint work with M. Griebel and Ch. Rieger

Meshfree quadrature [Schaback 2014], [Griebel, Rieger 2015], [Z. 2015], [Oettershagen 2017]

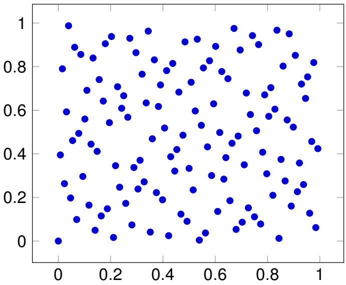
Quadrature rule

$$\int_\Gamma f(\boldsymbol{x})d\boldsymbol{x} \approx \sum_{i=1}^{N_\Gamma} \alpha_i f(\boldsymbol{x}_i)$$

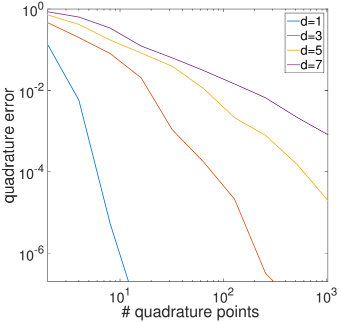
$$\boldsymbol{\alpha} = {A_{k,X}}^{-1}\boldsymbol{b}, \quad b_i = \int_\Gamma k(\boldsymbol{x}_i, \boldsymbol{x})d\boldsymbol{x}$$

up to exponential convergence with QMC points

Quadrature points $\boldsymbol{x}_i$



Convergence



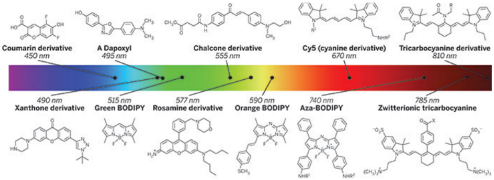
Machine learning in quantum chemistry

Objectives and challenges

- computational exploration of chemical compound space

Proposed solution

- machine learning: predicting energies of unknown molecules
- kernel ridge regression: $p(\boldsymbol{M}) = \sum_{i=1}^{N_\Gamma} \alpha_i k(\boldsymbol{M}, \boldsymbol{M}_i)$
  ($\boldsymbol{M}_i$ representation (e.g. coulomb matrix) of molecule)
- **multi-fidelity machine learning models**



SNF project by H. Harbrecht & A. v. Lilienfeld within 75 NRP Big Data National Research Programme

Outline

Motivation

Hierarchical matrices

Many-core parallelization of $\mathcal{H}$ matrices

Solvers on many-core clusters

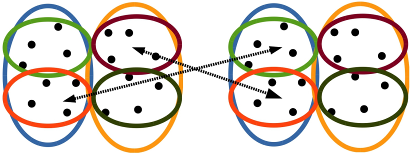
Fast solvers for collocation matrices

Potential choices of solvers

- direct factorization $\rightarrow O({N_\Gamma}^3)$
- iterative solvers $\rightarrow O(N_{iter} N_{MVP})$

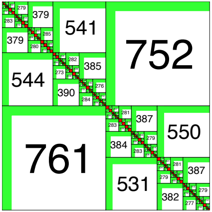
Fast MVP by hierarchical ($\mathcal{H}$) matrices [Hackbusch 1999],...

- matrix entries $k(\boldsymbol{y}_i, \boldsymbol{y}_j)$ corresponding to tuples of points

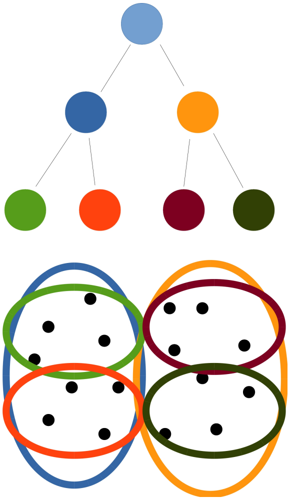


$$\begin{pmatrix} k(\boldsymbol{y}_1, \boldsymbol{y}_1) & \cdots & k(\boldsymbol{y}_1, \boldsymbol{y}_{N_\Gamma}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{y}_{N_\Gamma}, \boldsymbol{y}_1) & \cdots & k(\boldsymbol{y}_{N_\Gamma}, \boldsymbol{y}_{N_\Gamma}) \end{pmatrix}$$

- matrix approximation via tree-based point set decomposition
- approximation of subblocks if corresponding point sets are *far away* i.e. admissible

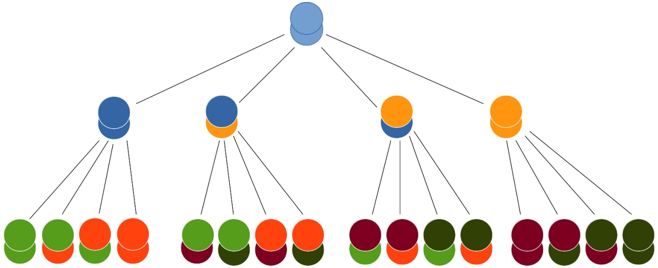


Cluster tree



- hierarchical decomposition of point set into clusters
- tree of subsets of the underlying point set
- splitting of subsets
  e.g. based on *cardinality based clustering* (CBC)
- implementation
  $\rightarrow$ **space filling curve**

Block cluster tree



- tree of subset / cluster tuples
- subset splitting based on cluster tree
- nodes representing subblocks of system matrix
- leaves either stored exactly or approximated if admissible
- admissibility condition:

$$\min\{\text{diam}(\Omega_\tau), \text{diam}(\Omega_\sigma)\} \leq \eta \text{dist}(\Omega_\tau, \Omega_\sigma)$$

fast MVP $\Leftrightarrow$ block tree traversal & leaf application

Matrix block approximation

Adaptive Cross Approximation [Bebendorf 2000]

- low-rank approximation
- algorithm (simplified):

For $r = 1, 2, \ldots, k$

$$\hat{\boldsymbol{u}}_r = A_{1:m, j_r} - \sum_{l=1}^{r-1} \boldsymbol{u}_l (\boldsymbol{v}_l)_{j_r},$$

$$\boldsymbol{u}_r = (\hat{\boldsymbol{u}}_{i_r})^{-1} \hat{\boldsymbol{u}}_r, \text{ with } |(\hat{\boldsymbol{u}}_r)_{i_r}| = \|\hat{\boldsymbol{u}}_r\|_\infty,$$

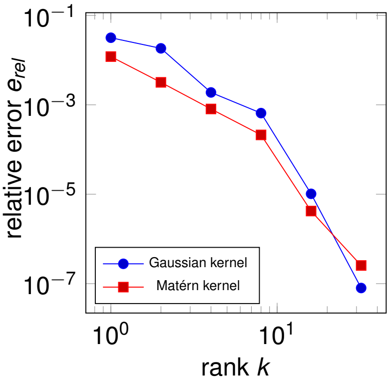
$$\boldsymbol{v}_r = (A_{i_r, 1:n})^\top - \sum_{l=1}^{r-1} (\boldsymbol{u}_l)_{i_r} \boldsymbol{v}_l$$

$$\text{if } \left(\|\boldsymbol{u}_r\|_2 \|\boldsymbol{v}_r\|_2 \leq \frac{\epsilon(1.0-\eta)}{1.0+\epsilon} \left\| \sum_{l=1}^{r} \boldsymbol{u}_l \boldsymbol{v}_l \right\|_F \right)$$

stop

- $A \approx \sum_{r=1}^{k} \boldsymbol{u}_r \boldsymbol{v}_r$

convergence of MVP ($d = 3$)



Outline

Motivation

Hierarchical matrices

Many-core parallelization of $\mathcal{H}$ matrices

Solvers on many-core clusters

Why targeting many-core hardware / clusters?

Top supercomputing systems

- China: **Tianhae-2**, Intel Xeon Phi 31S1P (Top 2)
- Europe: **Piz Daint**, Nvidia Tesla P100 (Top 3)
- US: **Titan**, Nvidia Tesla K20X (Top 4)
  $\Rightarrow$ **Summit** to come in 2018, Nvidia Volta architecture

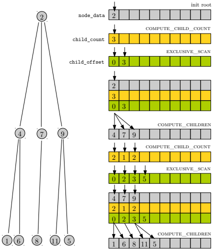
Machine learning

- *deep learning* often done on GPUs
- making kernel ridge regression available for many-core

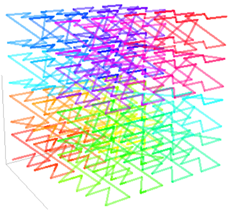
`hmglib`: Entirely GPU-based $\mathcal{H}$-matrix implementation

- Open Source library: LGPL, `github.com/zaspel/hmglib`
- **full $\mathcal{H}$ matrix construction (with ACA) and MVP on GPU**
- evaluation of $k(\cdot,\cdot)$ is cheap, i.e. control flow dominates
- preprint documenting details on arXive [Z., 2017c]

Core parts for high performance on many-core architecture

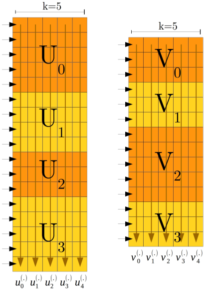


parallel tree traversal



source: Wikipedia

parallel spatial data structure



batching of similar small tasks

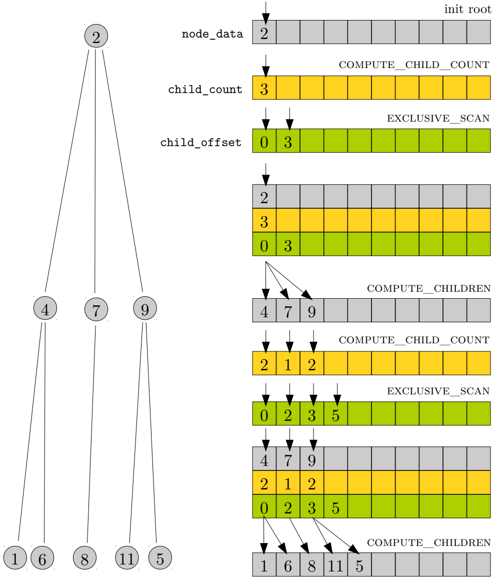
Parallel tree traversal

Idea

- old approach: array-based tree construction
- parallelization over tree level
- high degree of parallelism for wide trees

Implementation

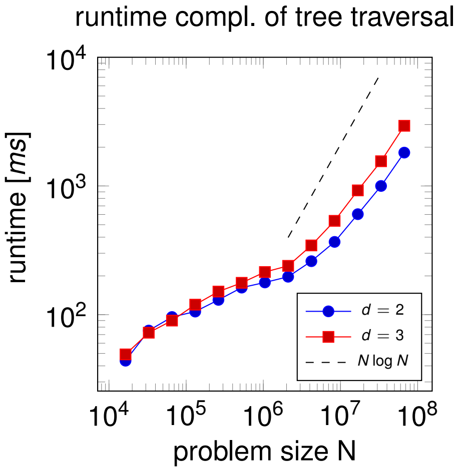
- complex algorithms (`EXCLUSIVE_SCAN`,...) from GPU library
- dynamic array allocation



Parallel tree traversal: Performance

Performance

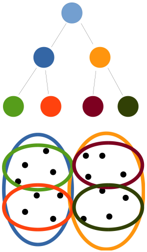
- measured for block cluster tree construction
- faster than $O(N \log N)$
- tree traversal negligible wrt. other $\mathcal{H}$ matrix components



Spatial data structure: Space filling curves

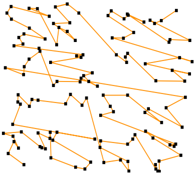
Hierarchical data structure: Cluster tree

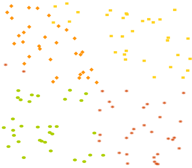
- decomposition of point set into clusters
- tree of subsets of the underlying points
- splitting of subsets e.g. based on *cardinality based clustering* (CBC)



Z-order curve as data structure

1. transformation of input point set X_Γ coordinates to Morton codes
2. sorting points following Morton codes
   $\Rightarrow$ neighboring points in list are close
3. splitting into point subsets of subsequent Morton codes
   $\Rightarrow$ **clustering strategy**



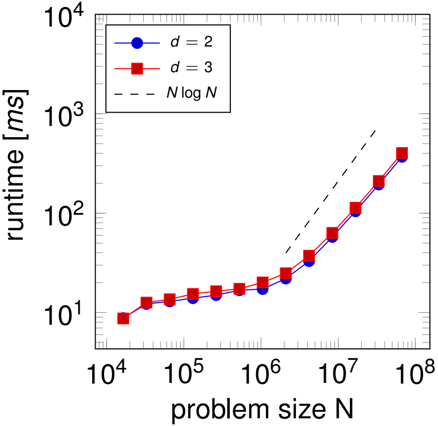


Spatial data structure: Performance

Performance

- measured for Morton code generation and parallel sorting
- Morton codes: $O(N)$
- parallel sorting: roughly $O(N \log N)$
- spatial data structure setup also negligible wrt. other $\mathcal{H}$ matrix components

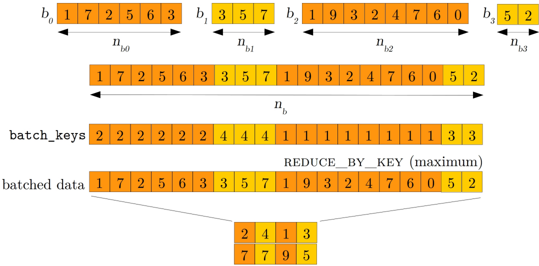
runtime compl. of spatial data struct.



Batching many similar operations on small data

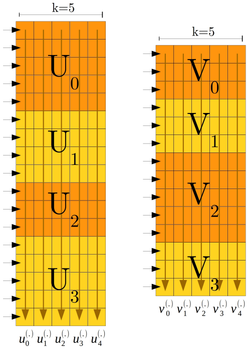
Frequent problem

- many identical operations on small data sets of different size
- parallelization over small data set *only*
  $\Rightarrow$ very inefficient

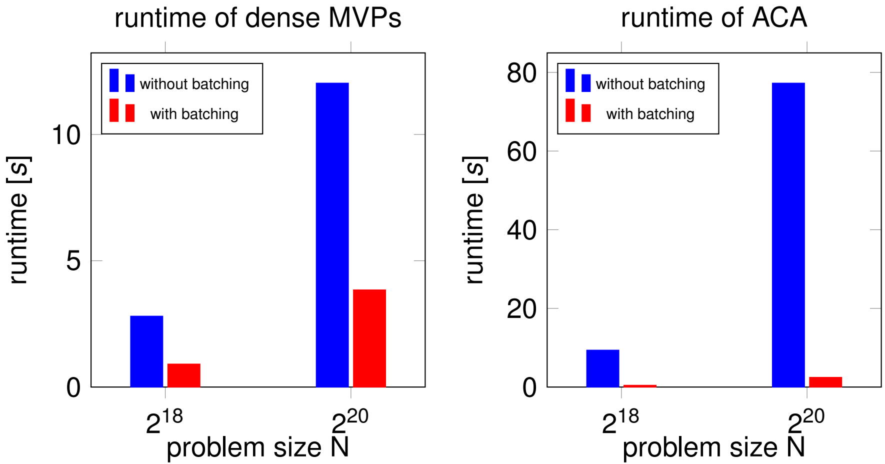


Solution options

1. mapping of data sets to different levels of parallelism of many-core processor
   $\Rightarrow$ complicated, hardware specific
2. **batching of data and use of key-based reductions from library**



Batching: Performance examples



Batching most important performance improvement in $\mathcal{H}$ matrix implementation.

hmglib: Complexity of $\mathcal{H}$ MVP

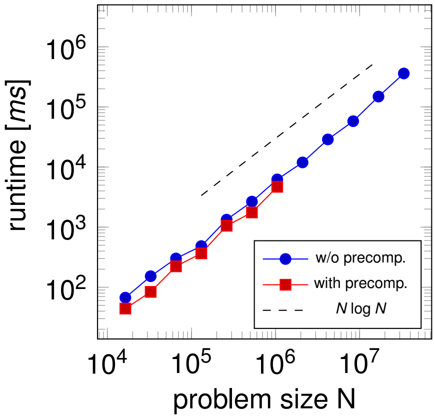
Objective

- getting high many-core parallel performance while keeping optimal complexity

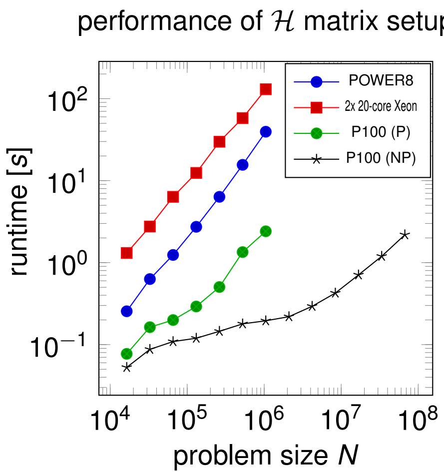
Result

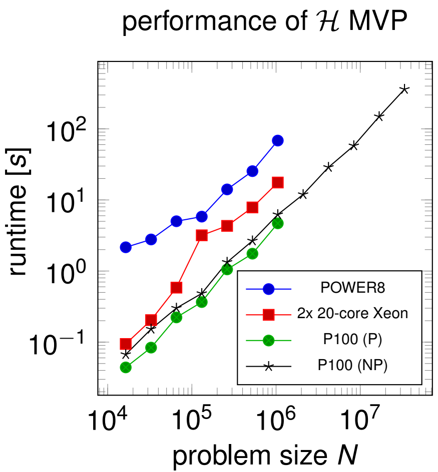
- fixed rank k
- **optimal $O(N \log N)$ complexity achieved**

runtime compl. of $\mathcal{H}$ MVP for $d = 2$



Total performance of `hmglib`





speedup in setup: **50x** speedup in matrix-vector product: **3x**

CPU performance is **multi-core** performance of `H2Lib`. Comparing roughly **equally priced** hardware: 2x20 core Xeon vs. Tesla P100

Outline

Motivation

Hierarchical matrices

Many-core parallelization of $\mathcal{H}$ matrices

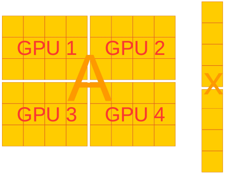
Solvers on many-core clusters

Krylov subspace solver for kernel linear system

MPLA

- **iterative dense linear solvers** for multi-GPU clusters
- runs on *Titan* at ORNL
- Open Source: LGPL, `github.com/zaspel/MPLA`
- $O(N_\Gamma{}^2)$ complexity matrix-vector products

Parallelization between GPUs

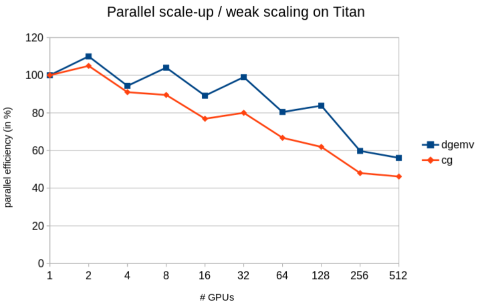


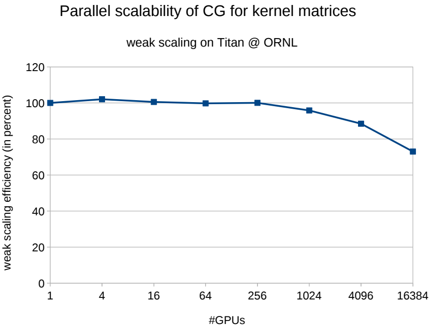
data exch. by CUDA-aware MPI

Parallelization on GPU

- kernel matrix setup written in CUDA
- use of CUBLAS for MVP
  $\Rightarrow$ BLAS impl. by vendor

Weak scalability results of pure Krylov solver on Titan





Matrix-based approach

- fill dense matrix in GPU memory
- apply BLAS `dgemv`
- problem: matrix size limited by GPU memory size

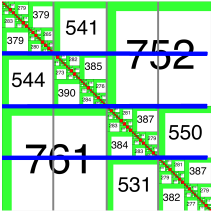
On-the-fly application

- sucessively generate and apply parts of the matrix on single GPU
- advantage: arbitrary size of matrix on GPU possible

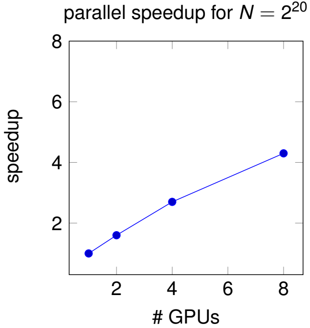
Parallelization by matrix decomposition

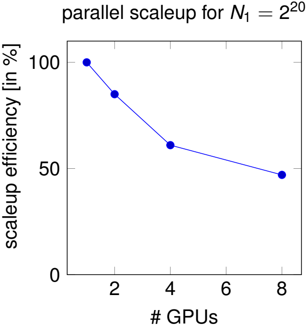
Implementation

- implementation by plugging **hmglib** into **MPLA** library
- matrix decomposition into row blocks



Scalability results





Summary

- solvers for dense systems in meshfree methods
- **hmglib** $\mathcal{H}$ matrix library runs in **MPLA**
- important applications in quadrature and machine learning

Literature

- Z., Algorithmic patterns for $\mathcal{H}$-matrices on many-core processors, eprint arXiv:1708.09707, 2017
- Z., Parallel RBF Kernel-Based Stochastic Collocation for Large-Scale Random PDEs, PhD thesis, University of Bonn, 2015.

Acknowledgements

- 75 NRP **Big Data** National Research Programme
- This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- The IBM POWER8 system with the NVIDIA Tesla P100 SXM2 used in the benchmarks for this research was donated by the NVIDIA PSG Cluster.

Summary

- solvers for dense systems in meshfree methods
- **hmglib** $\mathcal{H}$ matrix library runs in **MPLA**
- important applications in quadrature and machine learning

Literature

- Z., Algorithmic patterns for $\mathcal{H}$-matrices on many-core processors, eprint arXiv:1708.09707, 2017
- Z., Parallel RBF Kernel-Based Stochastic Collocation for Large-Scale Random PDEs, PhD thesis, University of Bonn, 2015.

Thank you!

Acknowledgements

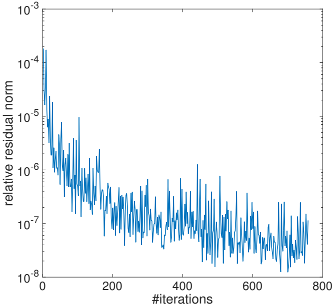
- 75 NRP **Big Data** National Research Programme
- This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- The IBM POWER8 system with the NVIDIA Tesla P100 SXM2 used in the benchmarks for this research was donated by the NVIDIA PSG Cluster.

Artificial test cases

- solving system for Gaussian kernel, manufactured RHS
- $N_\Gamma = 300\,000$ points of Halton sequence in $[0, 1]^d$

d=10

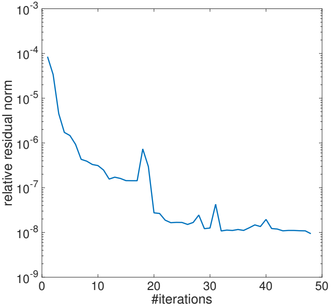
- 256 GPUs on Titan
- dense kernel matrix
- stopping: $\frac{\|\boldsymbol{r}_i\|}{\|\boldsymbol{b}\|} < 10^{-9}$



total runtime: ~3.65 minutes

d=2

- 1 GPU on Titan
- $\mathcal{H}$ MVP
- stopping: $\frac{\|\boldsymbol{r}_i\|}{\|\boldsymbol{b}\|} < 10^{-9}$



total runtime: ~3.65 minutes

Many-core $\mathcal{H}$-matrix implementations: Related work

H2Lib

- GPU-accelerated boundary element quadrature and $\mathcal{H}^2$-GCA compression

[Kriemann 2014]

- $\mathcal{H}$-LU factorization algorithms designed for many-core
- implemented on Xeon Phi
- strong emphasis on use of many-core architecture for *work part*

HiCMA: Hierarchical Computations on Manycore Architectures (Keyes et al.)

- seemingly very strong project towards hierarchical algorithms on many-core hardware
- unclear state, no (?) software freely available