Scalable solvers for meshless methods on many-core clusters

Peter Zaspel

University of Basel

QUIET 2017
SISSA, Trieste, July 18-21, 2017

Main objective for today

Solution of well-structured dense linear system

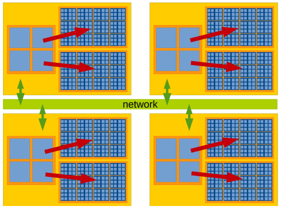
$$\begin{pmatrix} k(\boldsymbol{y}_1, \boldsymbol{y}_1) & \cdots & k(\boldsymbol{y}_1, \boldsymbol{y}_{N_\Gamma}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{y}_{N_\Gamma}, \boldsymbol{y}_1) & \cdots & k(\boldsymbol{y}_{N_\Gamma}, \boldsymbol{y}_{N_\Gamma}) \end{pmatrix} \boldsymbol{x} = \boldsymbol{b}$$

- $k : \Gamma \times \Gamma \to \mathbb{R}$ positive definite kernel function
- $\{\boldsymbol{y}_i\}_{i=1}^{N_\Gamma}$, $\boldsymbol{y}_i \in \mathbb{R}^d$ points, with N_Γ potentially **extremely** large

Applications

- uncertainty quantification
- quadrature
- machine learning

Fast and scalable solvers



Outline

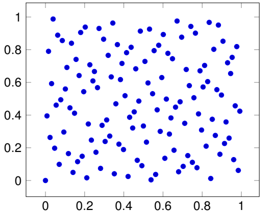
Outline

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

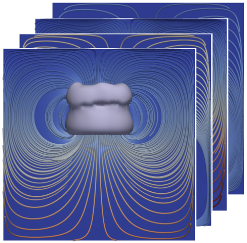
Example: Expectation value

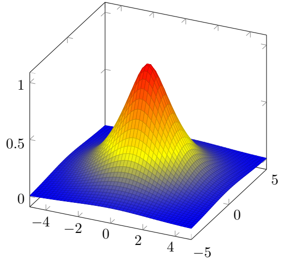
$$\mathbb{E}[u](\boldsymbol{x}) \approx \sum_{s=1}^{N_\Gamma} u(\boldsymbol{y}_s, \boldsymbol{x})\mathbb{E}[L_s] \approx \ldots = \sum_{s=1}^{N_\Gamma} u(\boldsymbol{y}_s, \boldsymbol{x}) \left((A_{k,X_\Gamma})^{-1} \boldsymbol{e} \right)_s$$

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



collocation points $\boldsymbol{y}_s$


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



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

Higher-order quadrature for QMC points

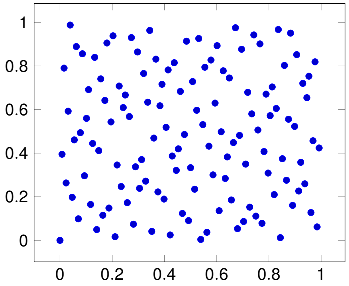
[Schaback 2014], [Griebel, Rieger 2015], [Z. 2015], [Oettershagen 2017]

Quadrature rule

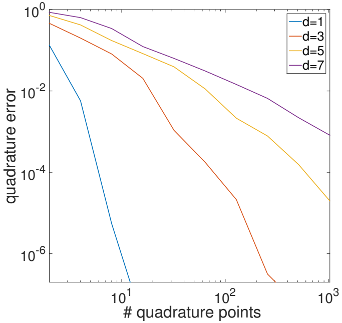
$$\int_\Gamma f(\boldsymbol{x}) d\boldsymbol{x} \approx \sum_{i=1}^{N_\Gamma} \alpha_i f(\boldsymbol{x}_i)$$

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

Quadrature points $\boldsymbol{x}_i$



Convergence (Gaussian)



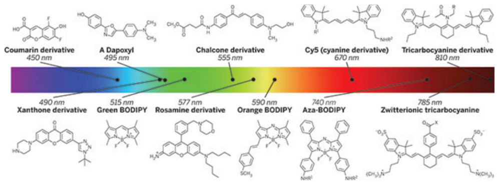
Machine learning in quantum chemistry

Objectives and challenges

- computational exploration of chemical compound space

Proposed solution

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



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

Outline

Scenarios

Scenario 1

- **original dense system has to be solved**
- example: unstructured, truely high-dimensional data sites
  $\Rightarrow$ no gain by e.g. low rank approx. in presymptotic regime
- (precond.) iterative Krylov subspace solvers $O(c(N_\Gamma)N_\Gamma^2)$

  $\Rightarrow$ $c(N_\Gamma) \equiv const$ for local Lagrange precond. on sphere

  (joint work w. M. Griebel, Ch. Rieger)

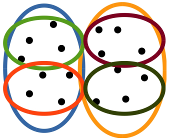
Scenario 2

- **dense matrix can be efficiently approximated**
- hierarchical matrices: $\sim O(N_\Gamma \log N_\Gamma)$ matrix-vector product
- use in Krylov subspace solver: $O(c(N_\Gamma)N_\Gamma \log N_\Gamma)$

Fast matrix-vector product by $\mathcal{H}$-Matrices

Hierarchical matrices [Hackbusch 1999],...

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

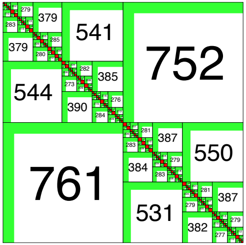


$$\begin{pmatrix} k(\boldsymbol{y}_1, \boldsymbol{y}_1) & \cdots & k(\boldsymbol{y}_1, \boldsymbol{y}_{N_\Gamma}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{y}_{N_\Gamma}, \boldsymbol{y}_1) & \cdots & k(\boldsymbol{y}_{N_\Gamma}, \boldsymbol{y}_{N_\Gamma}) \end{pmatrix}$$

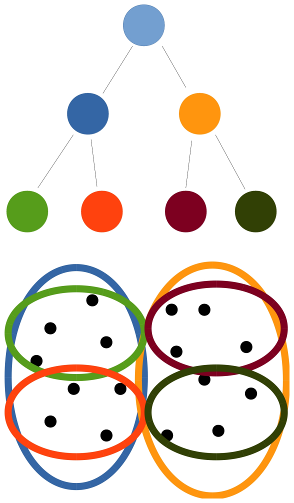
- matrix approximation via tree-based point set decomposition

- approximation of subblocks if corresponding point sets are *far away* i.e. admissible

- $\sim O(N_\Gamma \log N_\Gamma)$ complexity MVP

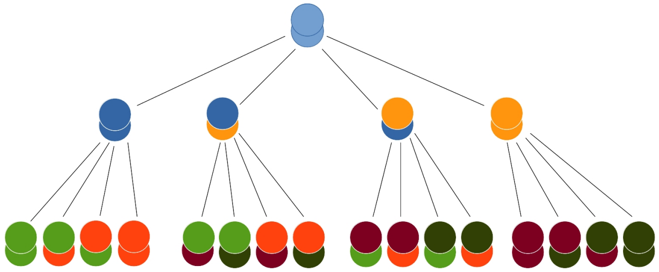


Cluster tree



- 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 (ACA) [Bebendorf 2000]

- low-rank approximation method
- algorithm (simplified):

```
For $r = 1, 2, \ldots, k$
    $\hat{\boldsymbol{u}}_r = A_{1:m,j_r} - \sum_{l=1}^{r-1} \boldsymbol{u}_l (\boldsymbol{v}_l)_{j_r}$,
    $\boldsymbol{u}_r = (\hat{\boldsymbol{u}}_{i_r})^{-1} \hat{\boldsymbol{u}}_r$, with $|(\hat{\boldsymbol{u}}_r)_{i_r}| = \|\hat{\boldsymbol{u}}_r\|_\infty$,
    $\boldsymbol{v}_r = (A_{i_r,1:n})^\top - \sum_{l=1}^{r-1} (\boldsymbol{u}_l)_{i_r} \boldsymbol{v}_l$
    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
```

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

Outline

Why targeting many-core clusters?

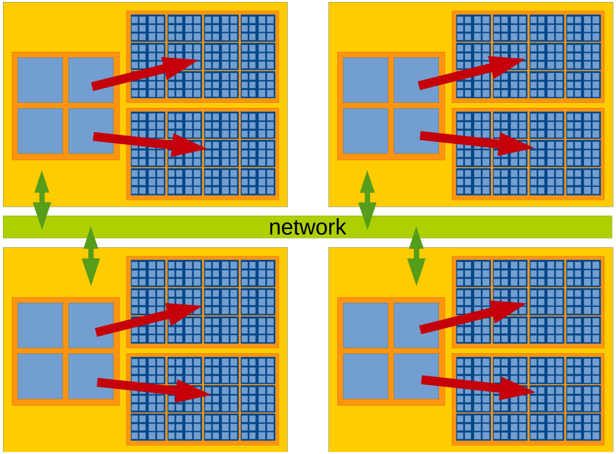
Top supercomputing systems

- 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

Challenge in top HPC systems



- special programming for many-core processors
- parallelization to get beyond a single many-core processor

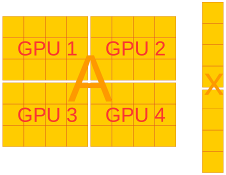
(Assumption: Want to compute on many-core procs., multi-core procs. for control)

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



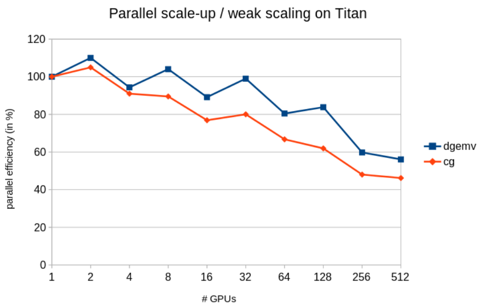
data exch. by CUDA-aware MPI

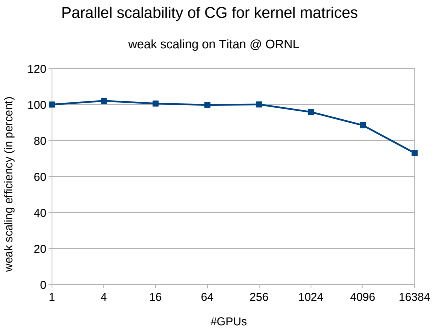
Parallelization on GPU

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

(localized preconditioner currently not part of the library)

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

Outline

Outline

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

Purely-GPU based $\mathcal{H}$-matrix implementation

hmglib

- Open Source library: LGPL, `github.com/zaspel/hmglib`
- Main objective: **Do everything on GPU.**

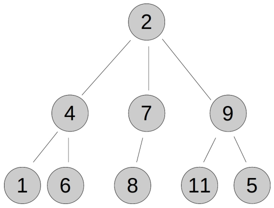
Algorithmic realization of fast matrix-vector product

- phase 1: setup
  - traversal of block tree
  - storage of all leaves ($\rightarrow$ dense MVP / ACA) in work queue
- phase 2: calculation
  - apply dense matrix-vector products
  - compute ACA / build dense matrix and apply results

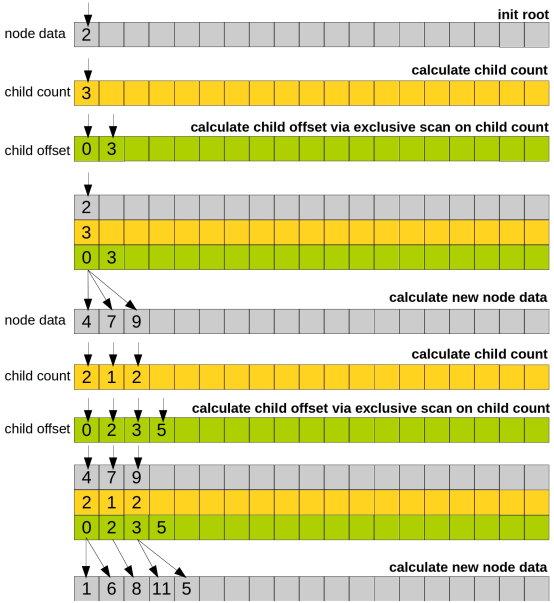
Phase 1: Components

1. general approach for tree traversal on GPU
2. spatial data structure for clustering
3. evaluation of admissibility condition *(skipped, $\rightarrow$ upcoming preprint)*
4. creation of work queue with leaves *(skipped, $\rightarrow$ upcoming preprint)*

Tree traversal on GPU



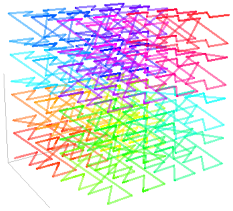
- reuse of old idea
  $\rightarrow$ tree construction in arrays
- wasting GPU performance
  $\rightarrow$ work queue approach?



Spatial data structure for cluster tree

Z-order curve / Morton codes

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**


source: Wikipedia

Implementation [Karras 2012]

- simple: point-wise Morton code computation by bit operations
- difficult: sorting following Morton codes $\Rightarrow$ `thrust`-library
- performance results on Nvidia Quadro K620 (29M pts in 3D)
  - compute codes: 98 ms
  - compute order: 640 ms
  - reorder 393 ms

Phase 2: Calculation

Example of work queue created during tree traversal

type: ACA ps 1: [0,3] ps 2: [4,7]	type: dense ps 1: [0,3] ps 2: [0,3]	type: ACA ps 1: [4,7] ps 2: [0,3]	type: dense ps 1: [4,7] ps 2: [4,7]	...

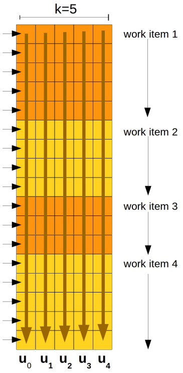
Batching of work items (ACA)

- increasing length of parallel vector by stacking several work items
- several ACAs done concurrently

$\Rightarrow$ similar idea for batching of dens MVP

Pros and cons

- full use of GPU processing units
- overhead for indexing, etc.



Performance results

	Quadro K620 time [s]			Tesla K20X time [s]		
N_Γ	Z-order	tree	MVP	Z-order	tree	MVP
2^{12}	0.0005	0.014	0.17	0.001	0.022	0.22
2^{13}	0.0006	0.019	0.30	0.001	0.032	0.27
2^{14}	0.0007	0.026	0.61	0.001	0.041	0.41
2^{15}	0.0009	0.045	1.35	0.001	0.067	0.72
2^{16}	0.0015	0.056	3.29	0.001	0.086	1.33
2^{17}	n/a	n/a	n/a	0.002	0.108	2.82
2^{18}	n/a	n/a	n/a	0.002	0.130	6.59
2^{19}	n/a	n/a	n/a	0.004	0.153	14.19

- Gaussian kernel, $X \subset [0,1]^2$, $\eta = 1$, $k = 16$
- $c_{leaf} = 512$ (Quadro K620), $c_{leaf} = 512$ (Tesla K20X)

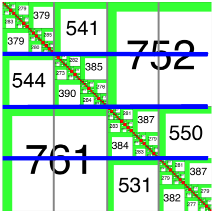
$\Rightarrow$ for now: ACA always recomputed due to memory limitation

Outline

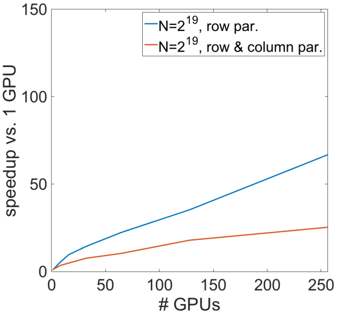
Parallelization by matrix decomposition

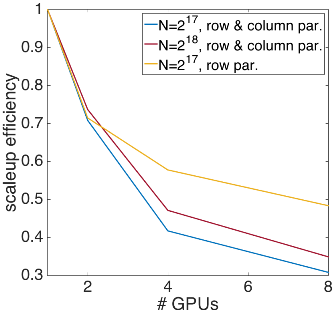
Implementation

- straight forward approach
- implementation by plugging **hmglib** into **MPLA** library



Scalability results

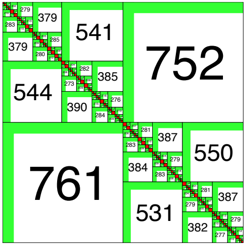


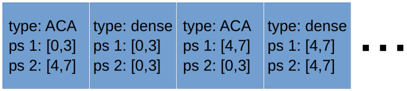


Outlook: Improving scalability results

Potential solution: Master - worker model

- use of work queue and task runtime prediction
- adaptive distribution of work items
- rather complex implementation





Outline

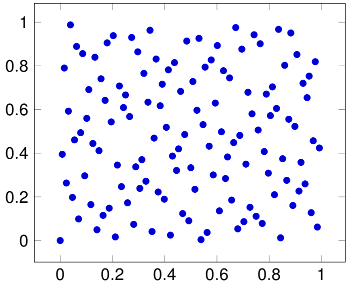
Remember: Meshfree quadrature [Schaback 2014], [Griebel, Rieger 2015], [Z. 2015], [Oettershagen 2017]

Quadrature rule

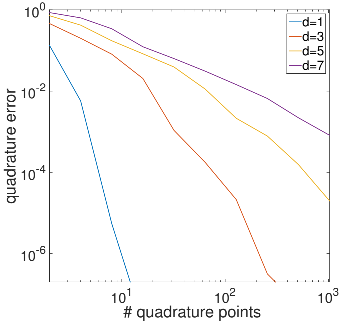
$$\int_\Gamma f(\boldsymbol{x}) d\boldsymbol{x} \approx \sum_{i=1}^{N_\Gamma} \alpha_i f(\boldsymbol{x}_i)$$

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

Quadrature points $\boldsymbol{x}_i$



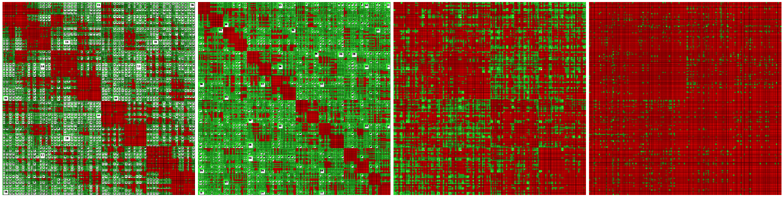
Convergence



Both scenarios within one application

Test with H2Lib

- approximation of system matrix for Gaussian kernel
- $\mathcal{H}$ matrix, ACA, $\epsilon = 10^{-5}$
- points: Halton sequence, Eucledian norm



$d = 3$ $d = 4$ $d = 5$ $d = 6$

Rough characterization of scenarios

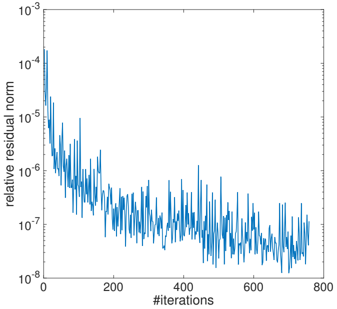
- $d \geq 5$: Scenario 1: Krylov with dense matrix
- $d < 5$: Scenario 2: Krylov solver with hierarchichal matrix

Artificial test cases

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

d=10

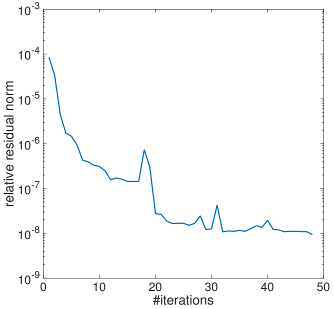
- 256 GPUs on Titan
- dense kernel matrix
- stopping: $\frac{\|\boldsymbol{r}_i\|}{\|\boldsymbol{b}\|} < 10^{-9}$



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: ∼26.5 minutes

Summary

- scalable dense kernel matrix solver
- **hmglib** $\mathcal{H}$ matrix library runs in **MPLA**
- important applications in quadrature and machine learning

Outlook

- using GPU with more memory for much faster $\mathcal{H}$ MVP
- improving scalability by different multi-GPU parallelization
- **preconditioners become crucial issue**

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.

Summary

- scalable dense kernel matrix solver
- **hmglib** $\mathcal{H}$ matrix library runs in **MPLA**
- important applications in quadrature and machine learning

Outlook

- using GPU with more memory for much faster $\mathcal{H}$ MVP
- improving scalability by different multi-GPU parallelization
- **preconditioners become crucial issue**

Thank you!

Acknowledgements

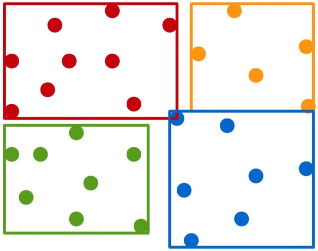
- 75 NRP **Big Data** National Research Programme
- This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Computation of admissibility condition

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

Bounding boxes

- use of bounding boxes of point subsets to approximate distance, diameter
- main challenge: computation of bounding boxes on each level of tree



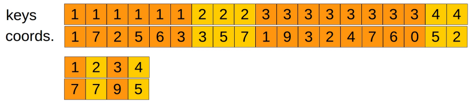
Computational task

- computation of min. / max. coordinates of many point subsets of different size
- subset sizes different on different levels of the tree

GPU parallelization of bounding box computation

Parallelization over coordinates

$\Rightarrow$ use of `reduce_by_key` in `thrust`

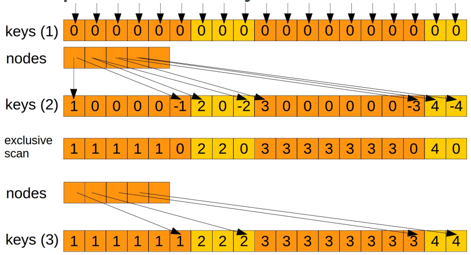


Performance results

level	time (1M p.)	time (4M p.)
0	17 ms	52 ms
1	24 ms	53 ms
2	23 ms	53 ms
3	19 ms	53 ms
4	21 ms	52 ms
5	22 ms	54 ms
6	18 ms	53 ms
7	22 ms	53 ms
8	22 ms	54 ms
9	24 ms	55 ms
10	27 ms	58 ms
11	21 ms	63 ms
12		78 ms
13		53 ms

(Matérn , $c_{leaf} = 1024$)

Computation of keys


keys (1)
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
nodes
keys (2)
1 0 0 0 0 -1 2 0 -2 3 0 0 0 0 0 0 -3 4 -4
exclusive scan
1 1 1 1 1 0 2 2 0 3 3 3 3 3 3 3 0 4 0
nodes
keys (3)
1 1 1 1 1 1 2 2 2 3 3 3 3 3 3 3 3 4 4