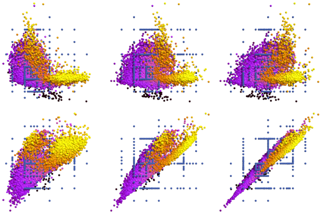
Numerical Data Mining with Sparse Grids at Extreme Scale

IPAM, Big Data meets Large-Scale Computing

Dirk Pflüger

Institute for Parallel and Distributed Systems / SimTech Cluster of Excellence, University of Stuttgart

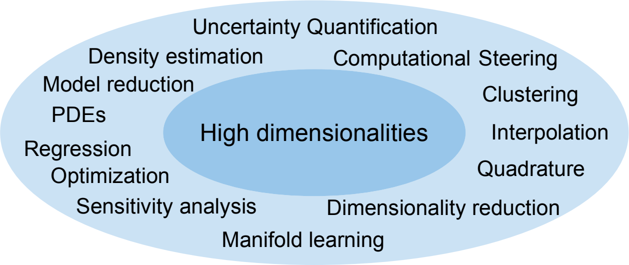
September 25, 2018



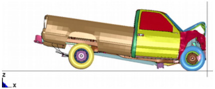
General Problem: High Dimensionalities

High dimensionalities

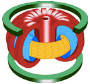
General Problem: High Dimensionalities

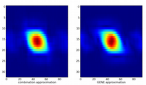


General Problem: High Dimensionalities

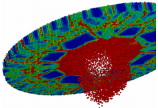


Automotive engineering

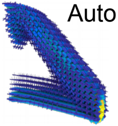




Plasma physics



Mechanics



Material sciences

Uncertainty Quantification

Density estimation

Computational Steering

Model reduction

Clustering

PDEs

High dimensionalities

Interpolation

Regression

Quadrature

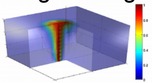
Optimization

Sensitivity analysis

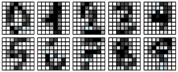
Dimensionality reduction

Manifold learning

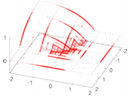
Hydraulic eingineering

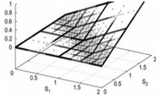


Data Mining

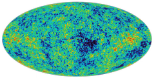


Financial mathematics

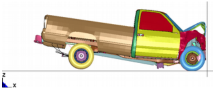




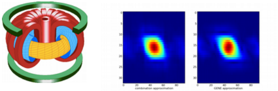
Astrophysics



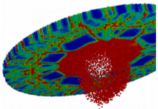
General Problem: High Dimensionalities



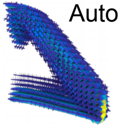
Automotive engineering



Plasma physics

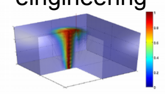


Mechanics



Material sciences

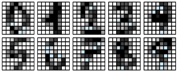
Hydraulic eingineering



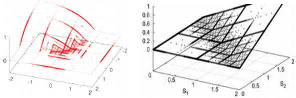
High dimensionalities

- Uncertainty Quantification
- Density estimation
- Computational Steering
- Model reduction
- Clustering
- PDEs
- Interpolation
- Regression
- Optimization
- Quadrature
- Sensitivity analysis
- Dimensionality reduction
- Manifold learning

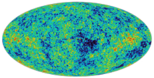
Data Mining



Financial mathematics



Astrophysics



Motivation: High Dimensionalities

Representation of higher-dimensional functions

- Required in plenty of applications:
  Quadrature, interpolation, PDEs, . . .
- Often as subtask
- Often implicitely given: no explicit/simple function formulation

Motivation: High Dimensionalities

Representation of higher-dimensional functions

- Required in plenty of applications:
  Quadrature, interpolation, PDEs, ...
- Often as subtask
- Often implicitely given: no explicit/simple function formulation

Conventional approach in low dimensions (≤ 4)

- Numerics, (spatial) discretization
- (Piecewise) polynomials, finite differences/elements/volumes, ...
- Typically functions of type

$$f \approx f_N(\vec{x}) := \sum_{i=1}^{N} \alpha_i \varphi_j(\vec{x})$$

Apply SciComp and HPC to DM

Curse of dimensionality

- Straightforward discretizations fail
- N grid points in $1D \Rightarrow N^d$ grid points in dD
- Therefore: Sparse Grids

Apply SciComp and HPC to DM

Curse of dimensionality

- Straightforward discretizations fail
- N grid points in $1D \Rightarrow N^d$ grid points in dD
- Therefore: Sparse Grids

Typical alternatives:

- Data dependent methods
  - Drawback: strong dependency on size/number of data points
- Stochastic methods
  - Drawback: no explicit function representation at hand

Apply SciComp and HPC to DM

Curse of dimensionality

- Straightforward discretizations fail
- N grid points in $1D \Rightarrow N^d$ grid points in dD
- Therefore: Sparse Grids

Typical alternatives:

- Data dependent methods
  - Drawback: strong dependency on size/number of data points
- Stochastic methods
  - Drawback: no explicit function representation at hand

Here: numerical data mining

- Take knowledge from SciComp and HPC
- Apply to Data Mining
- How parallel can we get?

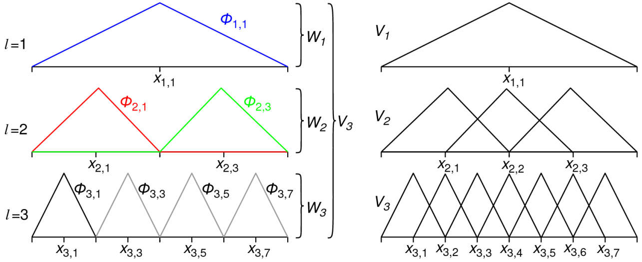
Overview

1. **Motivation: High Dimensionalities**
2. **Adaptive Sparse Grids to Counter the Curse**
3. **Numerical Data Mining and HPC**
   - Regression and Classification
   - Density Estimation
   - Clustering based on density estimation
4. **Summary**

Basic Idea: Hierarchical Basis

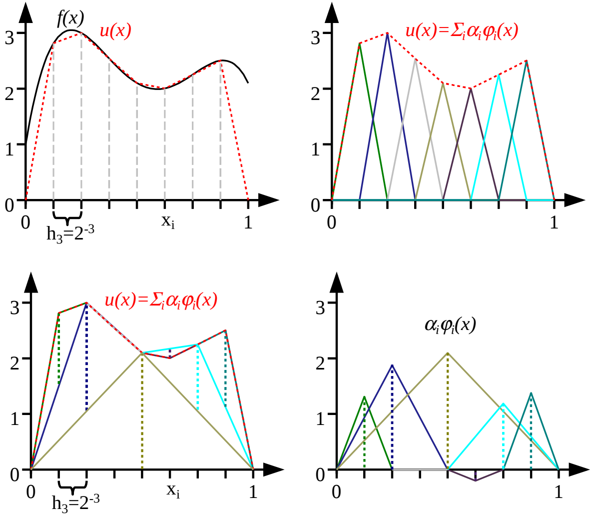
Hierarchical basis in 1d (here: piecewise linear)

$$f(x) = \sum_{l,i} \alpha_{l,i} \varphi_{l,i}(x)$$



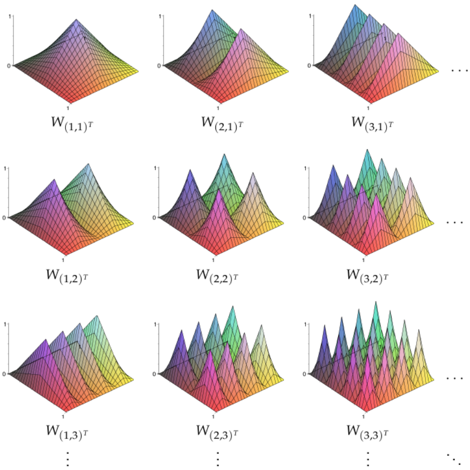
adaptive, incremental

Example: Interpolation 1d



Sparse Grids, Basic Idea (2)

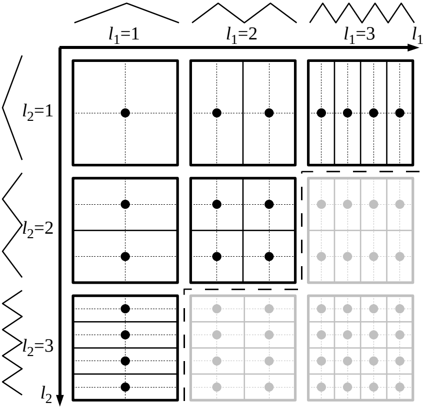
- Extension to d-dimensions via tensor product: $\varphi(\vec{x}) = \prod_{k=1}^{d} \varphi_k(x_k)$

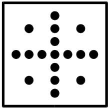


Sparse Grids

- Sparse grid space $V_n^{(1)}$:

$$V_n^{(1)} := \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}}$$

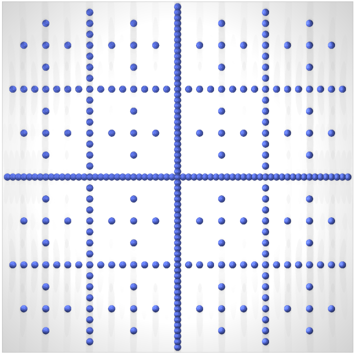


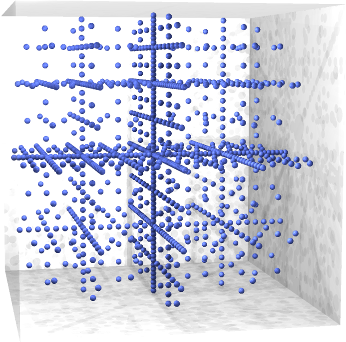


$V_3^{(1)}$

Sparse Grids, Examples

- Optimal selection of subspaces leads to sparse grid space $V_n^{(1)}$
- Examples for underlying sparse grids for level 6 in 2 and 3 dimensions



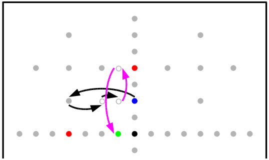


Sparse Grids – Properties

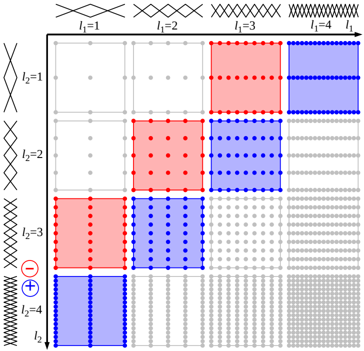
- For sufficiently smooth functions, i.e. $\left|D^{\vec{2}} f\right| = \left|\frac{\partial^{2d}}{\partial x_1^2 \cdots \partial x_d^2} f\right|$ bounded:
  - Order of magnitudes less costs (number of grid points)
  - Similar accuracy (interpolation, ellipt. PDEs)

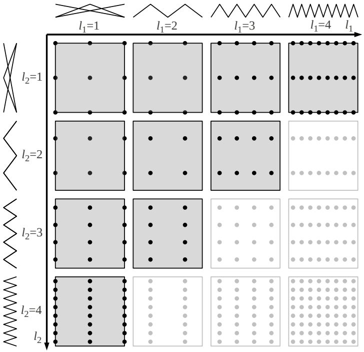
	full grid	sparse grids
costs	N^d	$N \log(N)^{d-1}$
error (L^2, L^∞)	h_n^2	$h_n^2 \log(N)^{d-1}$

- Higher dimensionalities feasible
- Adaptive approach
- Algorithms more complicated (multi-recursive)

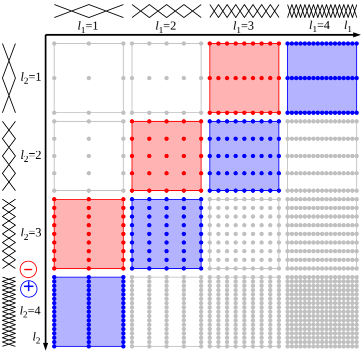


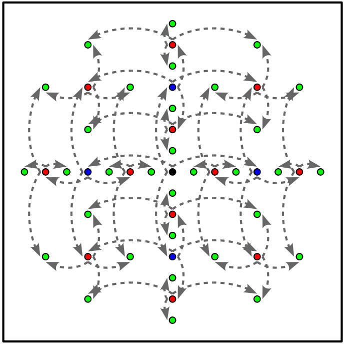
Combination Technique vs. Hierarchical Basis





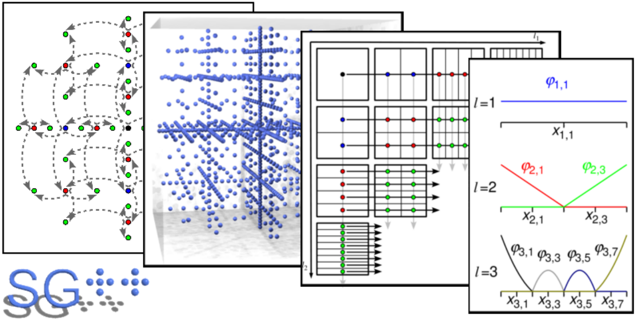
Combination Technique vs. Hierarchical Basis





SG^{++}

http://sgpp.sparsegrids.org



- General framework for (spatially adaptive) sparse grids
- Open source
- Active development
- Extensible (write/contribute your own module)

Overview

1. Motivation: High Dimensionalities
2. Adaptive Sparse Grids to Counter the Curse
3. Numerical Data Mining and HPC
   - Regression and Classification
   - Density Estimation
   - Clustering based on density estimation
4. Summary

Data-Driven: Classification and Regression

- Think of non-linear regression in several dimensions

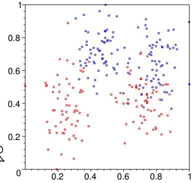
Data-Driven: Classification and Regression

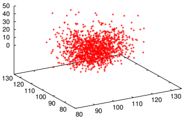
- Think of non-linear regression in several dimensions

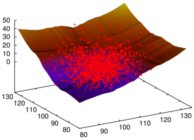
- Scattered data approximation problem:
  - Reconstruct unknown $f(\vec{x})$, represented by

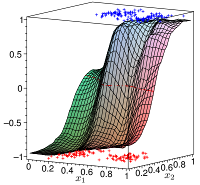
$$S = \{(\vec{x}_i, y_i) \in [0,1]^d \times T\}_{i=1}^{m}$$

  - Classification: $T = \{-1, +1\}$, regression: $T = \mathbb{R}$
  - Noisy data
  - Predict $f(\vec{x})$ for new datapoints $\vec{x}$
- Problems arise in finance, physics, ...









Regularization

- Ill-posed problem + deal with noise:
  - Thikonov regularization

$$\text{minimize } H[f] = \frac{1}{m}\sum_{i=1}^{m}(y_i - f(\vec{x}_i))^2 + \lambda \|f\|^2_{H^{\alpha}_{0,\mathrm{mix}}}$$

  - cost/error: ensure closeness to training data
  - Smoothness functional: close data $\rightarrow$ similar function value
  - trade-off via λ: regularization parameter

Regularization

- Ill-posed problem + deal with noise:
  - Thikonov regularization

$$\text{minimize } H[f] = \frac{1}{m}\sum_{i=1}^{m}(y_i - f(\vec{x}_i))^2 + \lambda \|f\|^2_{H^{\alpha}_{0,\mathrm{mix}}}$$

  - cost/error: ensure closeness to training data
  - Smoothness functional: close data $\rightarrow$ similar function value
  - trade-off via λ: regularization parameter

- Minimization leads to linear system

$$\left(\lambda C + B \cdot \frac{1}{m} B^T\right)\vec{\alpha} = \frac{1}{m} B\vec{y}$$

$$\text{with (simplest, e.g.)} \quad \begin{aligned} C &= I \\ B_{ij} &= \phi_i(\vec{x}_j) \end{aligned}$$

Introducing Numerics

- Classical approaches
  - SVM, NN, TPS, ... can be formulated with same approach
  - Are data dependent, typically $\Omega(m^2)$
- Data independent approach [Garcke, Griebel, Thess, 2001]
  - Introduce some degree of data-independence
  - Discretize space, apply ansatz functions associated to grid points
  - **Well-suited for vast datasets – only $\mathcal{O}(m)$**
  - Obtain explicit $f(\vec{x})$ (more informational value)

Introducing Numerics

- Classical approaches
  - SVM, NN, TPS, ... can be formulated with same approach
  - Are data dependent, typically $\Omega(m^2)$
- Data independent approach [Garcke, Griebel, Thess, 2001]
  - Introduce some degree of data-independence
  - Discretize space, apply ansatz functions associated to grid points
  - **Well-suited for vast datasets – only $\mathcal{O}(m)$**
  - Obtain explicit $f(\vec{x})$ (more informational value)
- Restrict to finite dimensional subspace V_N

$$\text{basis } \{\varphi_i(\vec{x})\}_{i=1}^N; \quad f_N(\vec{x}) = \sum_{i=1}^N \alpha_i \varphi_i(\vec{x})$$

- **Curse of dimensionality $\Rightarrow$ sparse grids**
- Solve linear system with CG, e.g.

Introducing Numerics

- Classical approaches
  - SVM, NN, TPS, ... can be formulated with same approach
  - Are data dependent, typically $\Omega(m^2)$
- Data independent approach [Garcke, Griebel, Thess, 2001]
  - Introduce some degree of data-independence
  - Discretize space, apply ansatz functions associated to grid points
  - **Well-suited for vast datasets – only $\mathcal{O}(m)$**
  - Obtain explicit $f(\vec{x})$ (more informational value)
- Restrict to finite dimensional subspace V_N

$$\text{basis } \{\varphi_i(\vec{x})\}_{i=1}^N; \quad f_N(\vec{x}) = \sum_{i=1}^N \alpha_i \varphi_i(\vec{x})$$

- **Curse of dimensionality $\Rightarrow$ sparse grids**
- Solve linear system with CG, e.g.

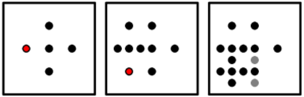
Large Data Sets and High Dimensions

Linear scaling in number of training data – $\mathcal{O}(m)$
Plus save in grid points:

Large Data Sets and High Dimensions

Linear scaling in number of training data – $\mathcal{O}(m)$
Plus save in grid points:

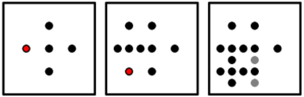
- Adaptive Refinement



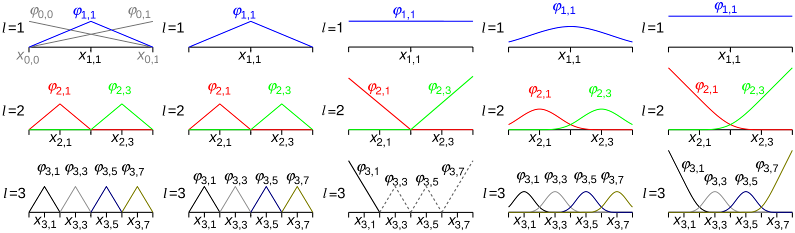
Large Data Sets and High Dimensions

Linear scaling in number of training data – $\mathcal{O}(m)$
Plus save in grid points:

- Adaptive Refinement



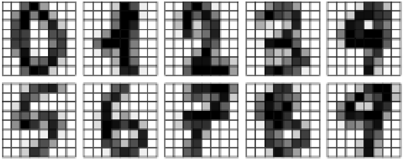
- Appropriate boundary treatment + suitable basis



Classification: Recognition of Handwritten Digits

- 8x8 pattern, $d = 64$
- 3823 training, 1797 testing
- One classifier for each class

$$\underset{j \in \{0,\ldots,9\}}{\arg\max}\ f_N^{(j)}(\vec{x})$$



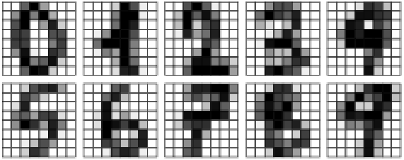
- Comparison with two benchmark studies [Devi02,Oliv.06]:

Classification method	Accuracy
k-NN	98.00%
Adaptive Sparse Grids	97.74%
RBF-DDA Networks	97.45%
Support Vector Machines	97.27%
AdaBoost	95.33%
MLP	89.05%

Classification: Recognition of Handwritten Digits

- 8x8 pattern, $d = 64$
- 3823 training, 1797 testing
- One classifier for each class

$$\arg\max_{j \in \{0,\ldots,9\}} f_N^{(j)}(\vec{x})$$

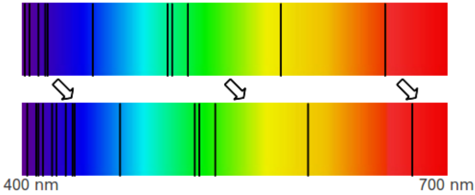


- Gridpoints per level for one of the classifiers (no boundary points)

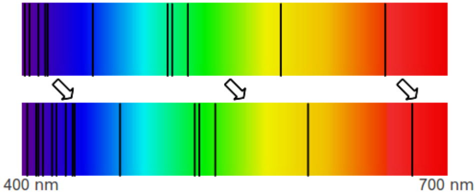
level ($\|l\|_1 - d + 1$)	full grids	regular sg	adaptive sg
1	1	1	1
2	728	128	128
3	116,920	8,324	993
4	11,272,976	366,080	510
5	876,113,056	12,263,680	128
$\sum$	887,503,681	12,638,213	1,760

Application: Cosmol. Redshift Estimation

Application: Cosmol. Redshift Estimation



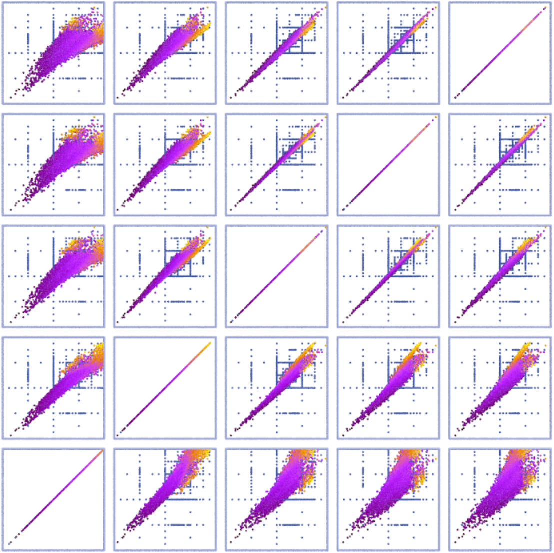
Application: Cosmol. Redshift Estimation



- Spectroscopic measurements expensive but accurate, photometric ones cheap
- Photometric measurements to predict spectroscopic parameters
- Five parameters (magnitudes: brightness measure for opt. filter)
- Large dataset: 430,000 data points; 60,000 for testing

Here:

- Refinement criteria critical
- Refining single grid point too greedy



Results: Big Data

- Best results training on whole dataset $(370,000/430,000)$

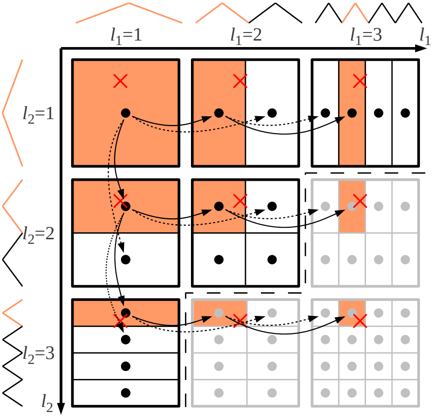
Method	σ_{rms}
CWW [csabai03]	0.0666
Bruzual-Charlot [csabai03]	0.0552
Interpolated spectra [csabai03]	0.0451
1^{rst}-nearest neighbors [csabai03]	0.0365
ClassX [suchkov05]	0.0340
Polynomial fit [csabai03]	0.0318
SVMs [wadadekar05]	0.027
Kd-tree [csabai03]	0.0254
Adaptive sparse grids	0.0220

Note: slightly different datasets

- Training time: 300,000s

Towards Efficient Software: Algorithms

- Solving iteratively $\left(\lambda m I + B \cdot B^T\right) \vec{\alpha} = B\vec{y}$
- Core: function evaluations: $(B^T \vec{\alpha})_i = f_N(\vec{x}_i)$
- Algorithmically efficient: $\mathcal{O}(\log(N)^d)$ for $\mathcal{O}(N \log(N)^{d-1})$ basis functions

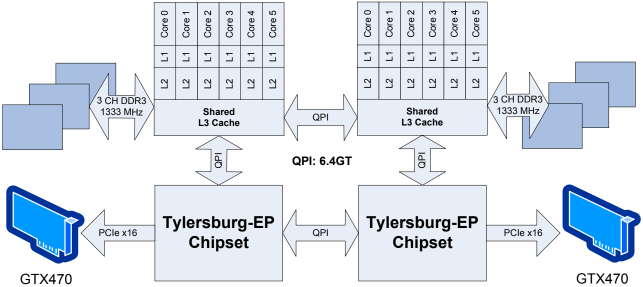


Hardware

Problem: modern hardware architectures

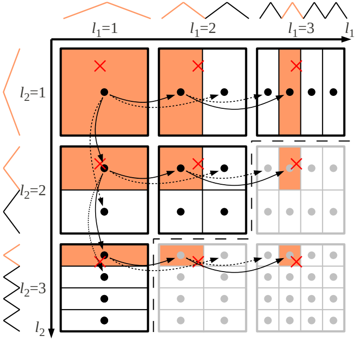
- Algorithm: multi-recursive, if statements, jumps in memory
- But: vector registers, pipelining, cache-hierarchy, accelerators, branch-divergence

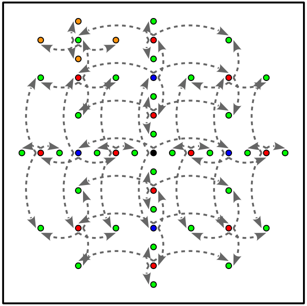
Westmere, 2x X5650, 2x GTX470



Hierarchical Basis and Data Structures

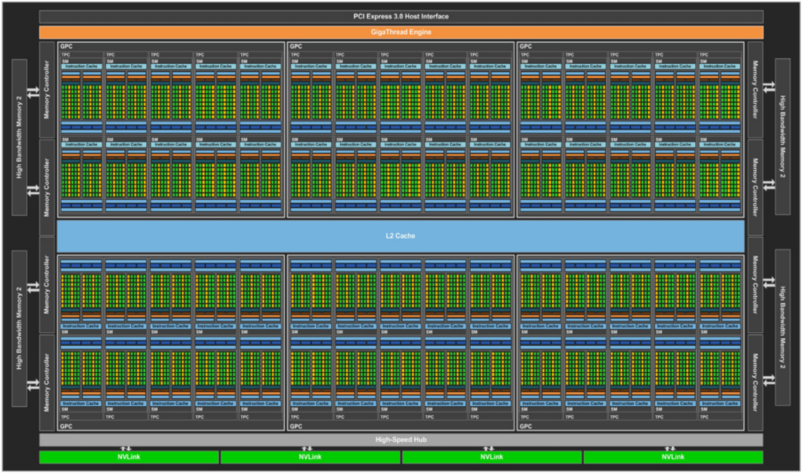
- Overlapping basis functions, tree-like data structures





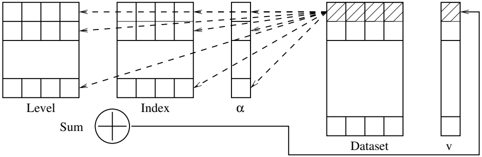
$$f(\vec{x}) = \sum_{\vec{l} \in \mathcal{L}} \sum_{\vec{i} \in \mathcal{I}_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \prod_{k=1}^{d} \varphi_{k_k,i_k}(x_k)$$

Efficient Hardware: e.g. P100



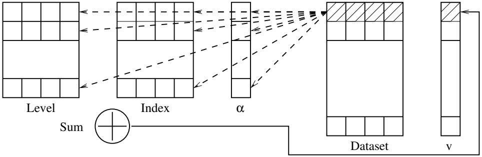
- Vast vector units: uniform computations required

Efficient Parallelization is Possible!

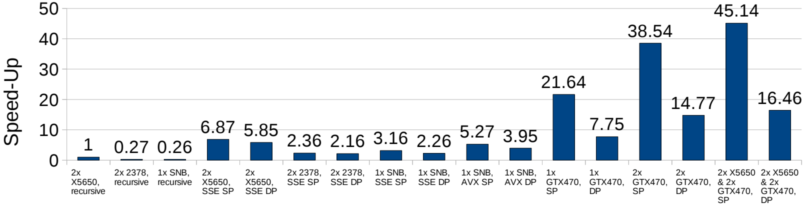


- Algorithmically inefficient approach: $\mathcal{O}(dN \log(N)^{d-1} \cdot m)$

Efficient Parallelization is Possible!

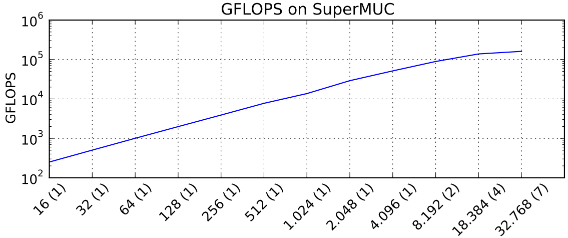


- Algorithmically inefficient approach: $\mathcal{O}(dN \log(N)^{d-1} \cdot m)$
- Redshift dataset ($410,000$ data points, $5d$)
- 6x 100 refined each, 16x more evaluations
- $\approx 300,000s \rightsquigarrow 15,800s \rightsquigarrow 350s$ (DP$\rightsquigarrow$SP)



Massively parallel Data Mining

- Data Mining shared and distributed on HPC:
  - SuperMUC, 32.768 cores: 3.6 s
  - 23% theoretical peak performance



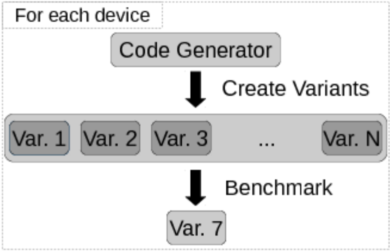
Hybrid Hardware – Auto-Tuning

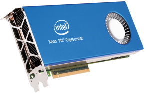
Current challenges:

- Hybrid Hardware, multiple vendors
- Inhomogeneous setting

Idea:

- Automatic tuning to hardware
- Code generation, OpenCL compute kernel

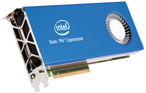




Hybrid Hardware – Auto-Tuning

Hardware platform	GFlops	Peak
Double Precision		
Nvidia K20X	771	50%
AMD FirePro W8100	1200	56%
Xeon Phi 31S1P	356	36%
4xXeon E7-8880v3 (72 Cores)	1016	46%
Single Precision		
Nvidia Titan X	3480	57%
2xGTX680, Tesla K10, Xeon Phi 31S1P	4417	35%

- 4d regression, lin. basis, adaptive
- Single CG iteration as benchmark



Numerical Experiments

Datasets

dataset	dim	size	$\|grid\|$
Chess4D	4	500,000	178,177
DR5	5	371,908	187,903
Friedman1	10	500,000	397,825

Scenario: regression

- realistic, adaptively refined grids
- 3 CG iterations
  $\approx 3.5 \cdot 10^6$ ($2.6 \cdot 10^6$) function evaluations
  plus scalar products etc.
- average over several runs
- auto-tuned OpenCL-code

Results

Chess4D

Hardware platform	[s]	GFlops	Peak
Double Precision			
4xXeon E7-8880v3 (72c)	15.0	1,000	38%
AMD FirePro W8100	10.9	1,378	63%
Nvidia Tesla K20X	19.3	777	59%
Xeon Phi 31S1P	34.6	433	43%
Single Precision			
4xXeon E7-8880v3 (72c)	5.5	2,735	52%
AMD FirePro W8100	5.9	2,558	61%
Nvidia Tesla K20X	7.5	2,000	51%
Xeon Phi 31S1P	17.7	844	42%
Geforce Titan X	4.2	3,573	51%

Results: Algorithmic Peak

Chess4D

Hardware platform	[s]	GFlops	Alg. Peak
Double Precision			
4xXeon E7-8880v3 (72c)	15.0	1,000	57%
AMD FirePro W8100	10.9	1,378	84%
Nvidia Tesla K20X	19.3	777	89%
Xeon Phi 31S1P	34.6	433	65%
Single Precision			
4xXeon E7-8880v3 (72c)	5.5	2,735	77%
AMD FirePro W8100	5.9	2,558	91%
Nvidia Tesla K20X	7.5	2,000	76%
Xeon Phi 31S1P	17.7	844	63%
Geforce Titan X	4.2	3,573	81%

Results: Algorithmic Peak

DR5

Hardware platform	[s]	GFlops	Alg. Peak
Double Precision			
4xXeon E7-8880v3 (72c)	15.2	966	55%
AMD FirePro W8100	10.9	1,350	82%
Nvidia Tesla K20X	18.5	793	91%
Xeon Phi 31S1P	35.1	418	62%
Single Precision			
4xXeon E7-8880v3 (72c)	5.4	2,730	77%
AMD FirePro W8100	5.9	2,469	88%
Nvidia Tesla K20X	7.7	1,897	72%
Xeon Phi 31S1P	17.1	858	64%
Geforce Titan X	3.8	3,895	88%

Results: Algorithmic Peak

Friedman1

Hardware platform	[s]	GFlops	Alg. Peak
Double Precision			
4xXeon E7-8880v3 (72c)	84.6	988	56%
AMD FirePro W8100	61.7	1,354	82%
Nvidia Tesla K20X	104.7	798	91%
Xeon Phi 31S1P	207.5	403	60%
Single Precision			
4xXeon E7-8880v3 (72c)	30.3	2,756	78%
AMD FirePro W8100	30.8	2,710	96%
Nvidia Tesla K20X	43.2	1,932	74%
Xeon Phi 31S1P	96.3	868	65%
Geforce Titan X	21.7	3,848	87%

Overview

Density Estimation

- Works quite well – can we do better, esp. for multi-class problems?
- And density estimation for data-driven UQ, etc.

Density Estimation

- Works quite well – can we do better, esp. for multi-class problems?
- And density estimation for data-driven UQ, etc.

New approach with **density estimation**

- Split data S into subsets S^k (one for each class)
- Estimate density functions f^k representing data S^k
- Get class prediction via

$$y = \arg\max_k f^k(\vec{x})$$

Density Estimation (2)

- Training set S^k for data in class k

$$S^k = \{\vec{x}_1, \ldots, \vec{x}_m\} \subset \mathbb{R}^d$$

  - Note: data unlabeled!

Density Estimation (2)

- Training set S^k for data in class k

$$S^k = \{\vec{x}_1, \ldots, \vec{x}_m\} \subset \mathbb{R}^d$$

  - Note: data unlabeled!

- Initial guess f_ϵ for density function f [Hegland et.al. 2000]:

$$f_\epsilon := \frac{1}{m} \sum_{i=1}^{m} \delta_{\vec{x}_i}$$

Density Estimation (2)

- Training set S^k for data in class k

$$S^k = \{\vec{x}_1, \ldots, \vec{x}_m\} \subset \mathbb{R}^d$$

  - Note: data unlabeled!

- Initial guess f_ϵ for density function f [Hegland et.al. 2000]:

$$f_\epsilon := \frac{1}{m} \sum_{i=1}^{m} \delta_{\vec{x}_i}$$

- Find estimated density function f as

$$f = \arg\min_{u \in V} \int_\Omega (u(\vec{x}) - f_\epsilon(\vec{x}))^2 \mathrm{d}\vec{x} + \lambda \|f\|^2_{H^\beta_{0,\mathrm{mix}}}$$

  - Note: regularizer does not preserve moments, but is sufficient and efficient!

Variational Formulation

- Again, sparse grids come into play
- With Ritz-Galerkin, we obtain

$$\langle f(\vec{x}), \varphi_j(\vec{x}) \rangle_{L^2} + \lambda \langle f(\vec{x}), \varphi_j(\vec{x}) \rangle_{H^{\beta}_{0,\mathrm{mix}}} = \frac{1}{m} \sum_{i=1}^{m} \varphi_j(\vec{x}_i), \qquad \forall \varphi_j$$

- Solve linear system

$$(P + \lambda I)\vec{\alpha} = \vec{b}$$

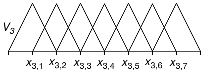
with

$$\begin{aligned} p_{ij} &= \langle \varphi_i(\vec{x}), \varphi_j(\vec{x}) \rangle_{L^2} \\ b_j &= \frac{1}{m} \sum_{i=1}^{m} \varphi_j(\vec{x}_i) \end{aligned}$$

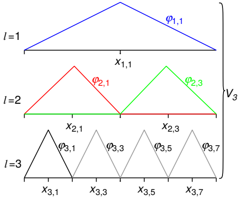
Again: Hierarchical Basis and Data Structures

- $P\vec{\alpha}$ problematic in hierarchical basis
- P dense!
- Consider 1D Gram matrix for $\langle \varphi_j(x), \varphi_k(x) \rangle_{L^2}$

$$
\begin{pmatrix}
1/24 & 1/48 & 0 & 0 & 0 & 0 & 0 \\
1/48 & 1/12 & 1/48 & 0 & 0 & 0 & 0 \\
0 & 1/48 & 1/12 & 1/48 & 0 & 0 & 0 \\
0 & 0 & 1/48 & 1/12 & 1/48 & 0 & 0 \\
0 & 0 & 0 & 1/48 & 1/12 & 1/48 & 0 \\
0 & 0 & 0 & 0 & 1/48 & 1/12 & 1/48 \\
0 & 0 & 0 & 0 & 0 & 1/48 & 1/24
\end{pmatrix}
$$



$$
\begin{pmatrix}
1/3 & 1/8 & 1/8 & 1/32 & 3/32 & 3/32 & 1/32 \\
1/8 & 1/6 & 0 & 1/16 & 1/16 & 0 & 0 \\
1/8 & 0 & 1/6 & 0 & 0 & 1/16 & 1/16 \\
1/32 & 1/16 & 0 & 1/12 & 0 & 0 & 0 \\
3/32 & 1/16 & 0 & 0 & 1/12 & 0 & 0 \\
3/32 & 0 & 1/16 & 0 & 0 & 1/12 & 0 \\
1/32 & 0 & 1/16 & 0 & 0 & 0 & 1/12
\end{pmatrix}
$$



- There are optimal complexity algorithms for mat-vec in $\mathcal{O}(N)$, but computationally inefficient...

Side-Note: Offline/Online Splitting

$$(P + \lambda I)\vec{\alpha} = \vec{b}$$

- Note: $(P + \lambda I)$ now independent of data!
- For repeated tasks: Offline/Online splitting possible!

www.simtech.uni-stuttgart.de

Side-Note: Offline/Online Splitting

$$(P + \lambda I)\vec{\alpha} = \vec{b}$$

- Note: $(P + \lambda I)$ now independent of data!
- For repeated tasks: Offline/Online splitting possible!
- Idea: LU decomposition of $(P + \lambda I)$
  - Problem: one has to fix λ

Side-Note: Offline/Online Splitting

$$(P + \lambda I)\vec{\alpha} = \vec{b}$$

- Note: $(P + \lambda I)$ now independent of data!
- For repeated tasks: Offline/Online splitting possible!
- Idea: LU decomposition of $(P + \lambda I)$
  - Problem: one has to fix λ
- Better alternative: eigendecomposition of P
  - Offline phase: eigendecomposition $P = USU^T$ (U orthonormal, S diagonal)
  - Online phase: compute in $\mathcal{O}(N^2)$

$$\vec{\alpha} = (P + \lambda I)^{-1}\vec{b} = (USU^T + \lambda UU^T)^{-1}\vec{b} = U(S + \lambda I)^{-1}U^T\vec{b}$$

- However: less suited for streaming approach and adaptivity

Results

Scenario

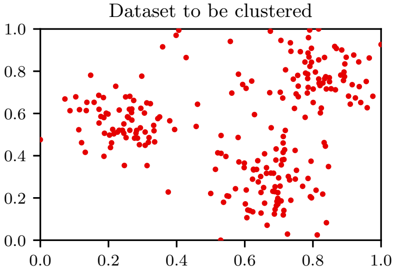
10d, 77,505 grid points (level 6), 10^8 data points

GFLOPS and achieved fraction of peak performance

	Tesla P100		FirePro W8100		2xXeon Gold 5120	
	float	double	float	double	float	double
right-hand side	3323 36%	1827 39%	1581 38%	839 40%	992 50%	538 55%
matrix-vector	2450 26%	1503 32%	969 23%	364 17%	337 17%	369 37%

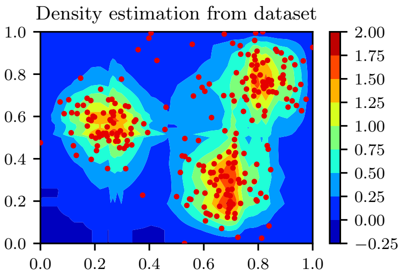
Overview

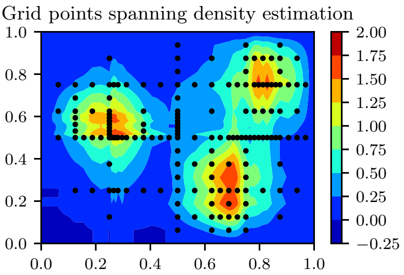
Clustering based on Density Estimation



- Ingredients: Density estimation, kNN, graph algorithms
- Based on [Peherstorfer,Pflüger,Bungartz 2012]

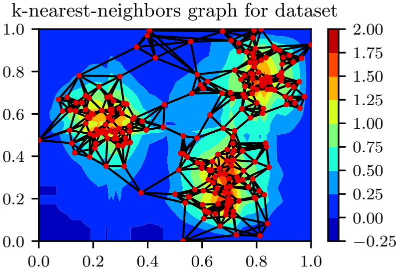
Clustering with Sparse Grids - Step I





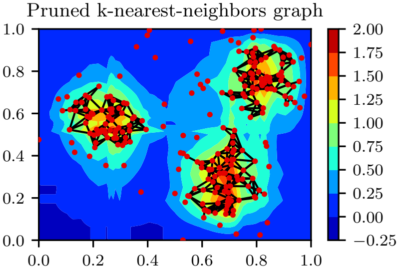
- Create density estimation of the grid points

Clustering with Sparse Grids - Step II



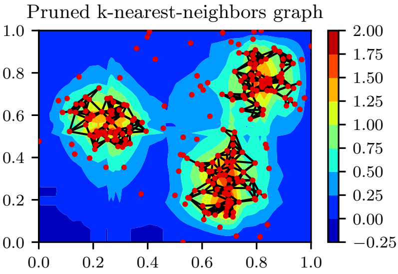
- Create k-nearest-neighbor graph for dataset

Clustering with Sparse Grids - Step III/IV



- Remove edges that intersect low-density region

Clustering with Sparse Grids - Step III/IV



- Remove edges that intersect low-density region
- Finally, find and return connected components as clusters

kNN Graph Creation and Pruning

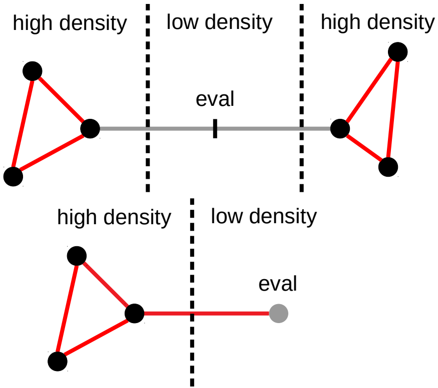
kNN graph creation:

- Classical $\mathcal{O}(m^2)$ kNN algorithm
- Large datasets enabled by fast implementation
- Parameter-free

kNN Graph Creation and Pruning

kNN graph creation:

- Classical $\mathcal{O}(m^2)$ kNN algorithm
- Large datasets enabled by fast implementation
- Parameter-free



Prune graph:

- Evaluate at data points
- Evaluate at midpoint of edges
- Prune if below threshold parameter

Compute Kernels

Algorithm 1: Right-hand side kernel

for $i = 0; i < N; i \leftarrow i + 1$ **do**
  for $j = 0; j < m; j \leftarrow j + 1$ **do**
    for $d = 0; d < ds; d \leftarrow d + 1$ **do**
      // 6 floating point ops

Algorithm 2: Create graph kernel

for $i = 0; i < m; i \leftarrow i + 1$ **do**
  for $j = 0; j < m; j \leftarrow j + 1$ **do**
    for $d = 0; d < ds; d \leftarrow d + 1$ **do**
      // 4 floating point ops

Algorithm 3: Density MV kernel

for $i = 0; i < N; i \leftarrow i + 1$ **do**
  for $j = 0; j < N; j \leftarrow j + 1$ **do**
    for $d = 0; d < ds; d \leftarrow d + 1$ **do**
      // 14 floating point ops

Algorithm 4: Prune graph kernel

for $i = 0; i < m; i \leftarrow i + 1$ **do**
  for $j = 0; j < N; j \leftarrow j + 1$ **do**
    for $d = 0; d < ds; d \leftarrow d + 1$ **do**
      // $6 \cdot (k + 1)$ fl. pt ops

- Streaming algorithms, fits to adaptively-refined sparse grids
- Parallelization of outermost loop

Algorithmic Properties

Kernel	FP ops.	Arith.Int. (loc=1)	Arith.Int. (loc=128)
d. right-hand side	$N \cdot m \cdot d \cdot 6$	$1.5\,\mathrm{F\,B}^{-1}$	$192\,\mathrm{F\,B}^{-1}$
d. matrix-vector	$N^2 \cdot d \cdot 14$	$1.2\,\mathrm{F\,B}^{-1}$	$149\,\mathrm{F\,B}^{-1}$
create kNN graph	$m^2 \cdot d \cdot 4$	$1.0\,\mathrm{F\,B}^{-1}$	$129\,\mathrm{F\,B}^{-1}$
prune kNN graph	$mN(k+1) \cdot d \cdot 6$	$4.5\,\mathrm{F\,B}^{-1}$	$576\,\mathrm{F\,B}^{-1}$

- Proper use of local memory necessary
- Modern hardware has machine balance of $\approx 10\,\mathrm{F\,B}^{-1}$

Node-Level Implementation

- OpenCL for (performance-)portability and vectorization
- Parameterized code generators for kernels:

Code generator	Generated code
see below	see below

Code generator

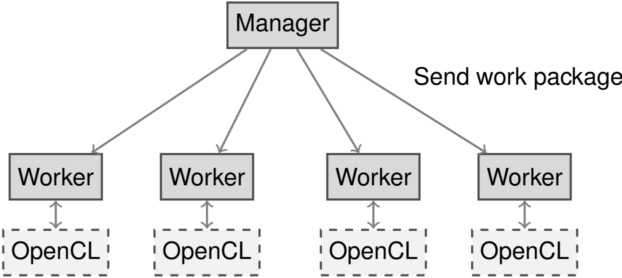
```
source << "__kernel mult("
       << "__global double *p) {\n";
if (useLocalMemory) {
source << "__local double v[5];\n";
for (int k = 0; k < 5; k += 1) {
  source << "v[" << k << "] = p[" << k
         << "];\n";
}
```

Generated code

```
__kernel mult(__global double *p)
__local double v[5];
v[0] = p[0];
v[1] = p[1];
v[2] = p[2];
v[3] = p[3];
v[4] = p[4];
```

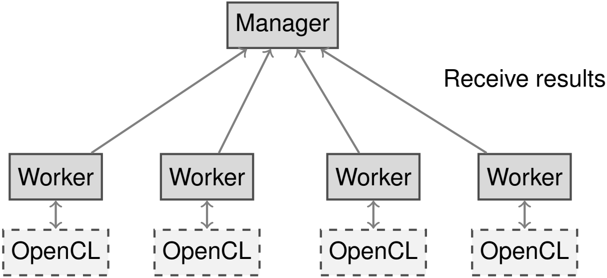
Distributed Approach

- Classical manager-worker scheme



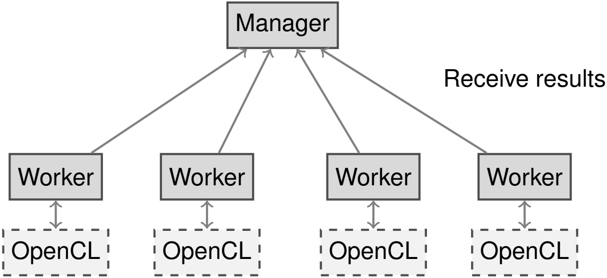
Distributed Approach

- Classical manager-worker scheme
- Work packages input start/stop indices, 1d result data



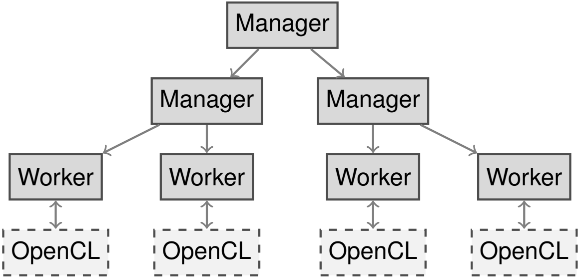
Distributed Approach

- Classical manager-worker scheme
- Work packages input start/stop indices, 1d result data
- Next work package transmitted during computation



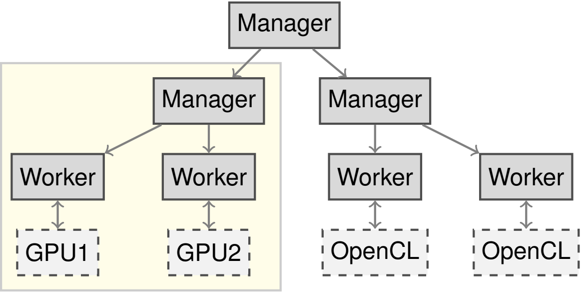
Distributed Approach

- Classical manager-worker scheme
- Work packages input start/stop indices, 1d result data
- Next work package transmitted during computation
- Network created according to configuration file



Distributed Approach

- Classical manager-worker scheme
- Work packages input start/stop indices, 1d result data
- Next work package transmitted during computation
- Network created according to configuration file
- Multi-GPU support on single node through MPI



Performance and Performance Portability

Scenario

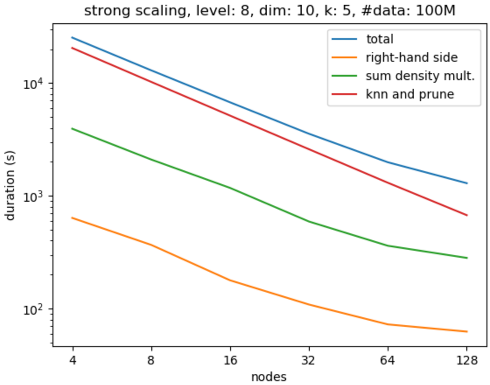
10d, 77,505 grid points (level 6), 10^8 data points

GFLOPS and achieved fraction of peak performance

	Tesla P100		FirePro W8100		2xXeon Gold 5120	
	float	double	float	double	float	double
dens. right-hand side	3323 36%	1827 39%	1581 38%	839 40%	992 50%	538 55%
dens. matrix-vector	2450 26%	1503 32%	969 23%	364 17%	337 17%	369 37%
create graph	4987 54%	3410 73%	1866 44%	832 40%	1124 57%	743 75%
prune graph	4626 50%	1859 40%	1616 38%	759 36%	940 48%	588 60%

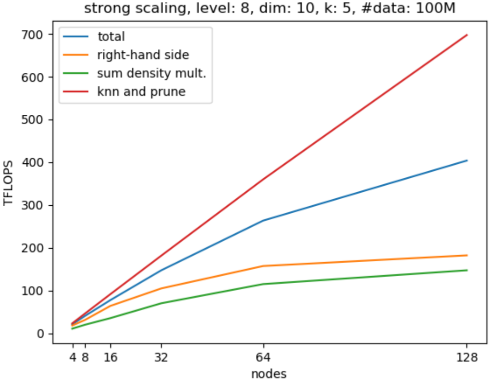
Strong Scaling on Piz Daint

- Strong scaling results, Cray XC50 Piz Daint, 1xTesla P100 per node
- 10D, level 8
- 10^8 data points, 5-NN
- $>$ 30% peak performance on 128 nodes!



Strong Scaling on Piz Daint

- Strong scaling results, Cray XC50 Piz Daint, 1xTesla P100 per node
- 10D, level 8
- 10^8 data points, 5-NN
- $>$ 30% peak performance on 128 nodes!



Ongoing and Future Work

- Larger datasets
- Approximate kNN
- Better data locality
- New hardware :-)
- Compression (mixed-precision arithmetics)
- . . .

Overview

Summary

SciComp for Data Mining

- high-dimensional approximation
- for moderately dimensional, numerical data
- linear scaling in data size: big data!

Summary

SciComp for Data Mining

- high-dimensional approximation
- for moderately dimensional, numerical data
- linear scaling in data size: big data!

Hierarchical and nested algorithms

- challenge for software and parallelization
- trading dofs against coupling
- sophisticated algorithms
- see also: `sgpp.sparsegrids.org`

Summary

SciComp for Data Mining

- high-dimensional approximation
- for moderately dimensional, numerical data
- linear scaling in data size: big data!

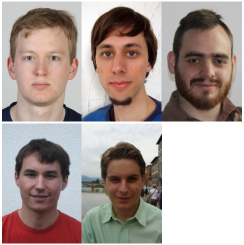
Hierarchical and nested algorithms

- challenge for software and parallelization
- trading dofs against coupling
- sophisticated algorithms
- see also: `sgpp.sparsegrids.org`

High-performance computing

- trade complexity against speed
- auto-tuning to optimize for hardware
- data mining at extreme scale

Thanks to:



. . . and all others!

SimTech Cluster of Excellence
Universität Stuttgart

Thank you for your interest!

D. Pfander, M. Brunn, and D. Pflüger.
AutoTuneTMP: Auto-Tuning in C++ With Runtime Template Metaprogramming.
In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1123–1132, 2018.

Fabian Franzelin and Dirk Pflüger.
Limiting ranges of function values of sparse grid surrogates.
In *Sparse Grids and Applications - Miami 2016*, pages 69–91, Cham, 2018. Springer International Publishing.

David Pfander, Alexander Heinecke, and Dirk Pflüger.
A new subspace-based algorithm for efficient spatially adaptive sparse grid regression, classification and multi-evaluation.
In *Sparse Grids and Applications - Stuttgart 2014*, pages 221–246. Springer International Publishing, 2016.

Alexander Heinecke, Roman Karlstetter, Dirk Pflüger, and Hans-Joachim Bungartz.
Data mining on vast data sets as a cluster system benchmark.
Concurrency and Computation: Practice and Experience, 2015.

Benjamin Peherstorfer, Dirk Pflüger, and Hans-Joachim Bungartz.
Clustering based on density estimation with sparse grids.
In *KI 2012: Advances in Artificial Intelligence*, volume 7526 of *Lecture Notes in Computer Science*. Springer, October 2012.

Dirk Pflüger.
Spatially Adaptive Sparse Grids for High-Dimensional Problems.
Verlag Dr. Hut, München, February 2010.