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

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

- Automotive engineering
- Plasma physics
- Mechanics
- Financial mathematics
- Astrophysics
- Material sciences
- Hydraulic engineering
- Data Mining
- Uncertainty Quantification
- Computational Steering
- Clustering
- Interpolation
- Quadrature
- Density estimation
- Model reduction
- Regression
- Optimization
- Sensitivity analysis
- Dimensionality reduction
- Manifold learning
- Regression
- Density estimation
- PDEs
- Mechanics
- Quadrature

Dirk Pflüger: Numerical Data Mining with Sparse Grids at Extreme Scale
IPAM, Big Data meets Large-Scale Computing, September 25, 2018
General Problem: High Dimensionalities

- Automotive engineering
- Plasma physics
- Data Mining
- Mechanics
- Material sciences
- Hydraulic engineering
- Financial mathematics
- Astrophysics

Related topics:
- Uncertainty Quantification
- Computational Steering
- Clustering
- Interpolation
- Quadrature
- Density estimation
- Model reduction
- PDEs
- Regression
- Optimization
- Sensitivity analysis
- Dimensionality reduction
- Manifold learning
- Regression
- Density estimation
- Material sciences
- Mechanics
- Astrophysics
Motivation: High Dimensionalities

Representation of higher-dimensional functions

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

Conventional approach in low dimensions ($\leq 4$)

- 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

\[ f(x) = u(x) = \sum_i \alpha_i \phi_i(x) \]

\[ h_3 = 2^{-3} \]

Dirk Pflüger: Numerical Data Mining with Sparse Grids at Extreme Scale
IPAM, Big Data meets Large-Scale Computing, September 25, 2018
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** $V_n^{(1)}$:

$$V_n^{(1)} := \bigoplus_{|\vec{i}|_1 \leq n+d-1} W_{\vec{i}}$$
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. \( |D^2 f| = \left| \frac{\partial^{2d}}{\partial x_1^2 \cdots \partial x_d^2} f \right| \) bounded:
  - Order of magnitudes less costs (number of grid points)
  - Similar accuracy (interpolation, ellipt. PDEs)

<table>
<thead>
<tr>
<th></th>
<th>full grid</th>
<th>sparse grids</th>
</tr>
</thead>
<tbody>
<tr>
<td>costs</td>
<td>( N^d )</td>
<td>( N \log(N)^{d-1} )</td>
</tr>
<tr>
<td>error ((L^2, L^\infty))</td>
<td>( h_n^2 )</td>
<td>( h_n^2 \log(N)^{d-1} )</td>
</tr>
</tbody>
</table>

- Higher dimensionalities feasible
- Adaptive approach
- Algorithms more complicated (multi-recursive)
Combination Technique vs. Hierarchical Basis

Dirk Pflüger: Numerical Data Mining with Sparse Grids at Extreme Scale
IPAM, Big Data meets Large-Scale Computing, September 25, 2018
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

  \[
  \text{minimize} \quad H[f] = \frac{1}{m} \sum_{i=1}^{m} (y_i - f(\vec{x}_i))^2 + \lambda \|f\|_{\mathcal{H}_0^{\alpha, \text{mix}}}^2
  \]

- cost/error: ensure closeness to training data
- Smoothness functional: close data \(\rightarrow\) similar function value
- trade-off via \(\lambda\): regularization parameter
Regularization

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

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

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

- Minimization leads to linear system

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

with (simplest, e.g.)
\[
C = I \\
B_{ij} = \phi_i(\vec{x}_j)
\]
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 $O(m)$**
  - 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 $O(m)$
  - 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 $O(m)$
  - 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 – $O(m)$

Plus save in grid points:
Large Data Sets and High Dimensions

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

- Adaptive Refinement

![Diagram showing adaptive refinement process]
Large Data Sets and High Dimensions

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

- Adaptive Refinement

  ![Adaptive Refinement Diagram]

- Appropriate boundary treatment + suitable basis

![Boundary Treatment Diagram]
Classification: Recognition of Handwritten Digits

- 8x8 pattern, \( d = 64 \)
- 3823 training, 1797 testing
- One classifier for each class
  \[
  \arg \max_{j \in \{0,\ldots,9\}} f_N^{(j)}(\tilde{x})
  \]

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

<table>
<thead>
<tr>
<th>Classification method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-NN</td>
<td>98.00%</td>
</tr>
<tr>
<td><strong>Adaptive Sparse Grids</strong></td>
<td><strong>97.74%</strong></td>
</tr>
<tr>
<td>RBF-DDA Networks</td>
<td>97.45%</td>
</tr>
<tr>
<td>Support Vector Machines</td>
<td>97.27%</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>95.33%</td>
</tr>
<tr>
<td>MLP</td>
<td>89.05%</td>
</tr>
</tbody>
</table>
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

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

Refinement criteria critical

Refining single grid point too greedy
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
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)

<table>
<thead>
<tr>
<th>Method</th>
<th>( \sigma_{\text{rms}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWW [csabai03]</td>
<td>0.0666</td>
</tr>
<tr>
<td>Bruzual-Charlot [csabai03]</td>
<td>0.0552</td>
</tr>
<tr>
<td>Interpolated spectra [csabai03]</td>
<td>0.0451</td>
</tr>
<tr>
<td>1(^{\text{rst}})-nearest neighbors [csabai03]</td>
<td>0.0365</td>
</tr>
<tr>
<td>ClassX [suchkov05]</td>
<td>0.0340</td>
</tr>
<tr>
<td>Polynomial fit [csabai03]</td>
<td>0.0318</td>
</tr>
<tr>
<td>SVMs [wadadekar05]</td>
<td>0.027</td>
</tr>
<tr>
<td>Kd-tree [csabai03]</td>
<td>0.0254</td>
</tr>
<tr>
<td>Adaptive sparse grids</td>
<td>0.0220</td>
</tr>
</tbody>
</table>

*Note: slightly different datasets*

- **Training time**: 300,000s
Towards Efficient Software: Algorithms

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

Diagram:
- Various grid structures with indexed levels $l_1$ and $l_2$.
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

Tylersburg-EP Chipset

GTX470

PCIe x16

QPI: 6.4GT

QPI

GTX470

PCIe x16

3 CH DDR3 1333 MHz

Shared L3 Cache

Core 0

L1

L2

Core 1

L1

L2

Core 2

L1

L2

Core 3

L1

L2

Core 4

L1

L2

Core 5

L1

L2

QPI

QPI

GTX470

PCIe x16

3 CH DDR3 1333 MHz

Shared L3 Cache

Core 0

L1

L2

Core 1

L1

L2

Core 2

L1

L2

Core 3

L1

L2

Core 4

L1

L2

Core 5

L1

L2
Hierarchical Basis and Data Structures

- Overlapping basis functions, tree-like data structures

\[ f(\bar{x}) = \sum_{\bar{l} \in \mathcal{L}} \sum_{\bar{i} \in \mathcal{I}_{\bar{l}}} \prod_{k=1}^{d} \varphi_{k, i_k}(x_k) \]
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: $O(dN \log(N)^{d-1} \cdot m)$
- Redshift dataset (410,000 data points, $5d$)
- 6x 100 refined each, 16x more evaluations
- $\approx 300,000 \text{s} \rightsquigarrow 15,800 \text{s} \rightsquigarrow 350 \text{s}$ (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

For each device:

1. Code Generator
2. Create Variants
   - Var. 1
   - Var. 2
   - Var. 3
   - ...  
   - Var. N
3. Benchmark
   - Var. 7
Hybrid Hardware – Auto-Tuning

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

- 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
  \[ \approx 3.5 \cdot 10^6 \text{ (2.6} \cdot 10^6 \text{)} \text{ function evaluations} \]
  plus scalar products etc.
- average over several runs
- auto-tuned OpenCL-code
## Results

### Chess4D

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

## Chess4D

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

### DR5

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

#### Friedman1

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

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

- Note: data unlabeled!
Density Estimation (2)

- Training set $S^k$ for data in class $k$
  \[ S^k = \{ \bar{x}_1, \ldots, \bar{x}_m \} \subset \mathbb{R}^d \]
  - Note: data unlabeled!

- Initial guess $f_e$ for density function $f$ [Hegland et.al. 2000]:
  \[ f_e := \frac{1}{m} \sum_{i=1}^{m} \delta_{\bar{x}_i} \]
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 $f_\epsilon$ 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
  \[ f = \arg \min_{u \in V} \int_{\Omega} (u(\vec{x}) - f_\epsilon(\vec{x}))^2 d\vec{x} + \lambda \| f \|_{H_0^{\beta, \text{mix}}}^2 \]

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

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

\[
\langle f(\vec{x}), \varphi_j(\vec{x}) \rangle_{L^2} + \lambda \langle f(\vec{x}), \varphi_j(\vec{x}) \rangle_{H^\beta_{0, \text{mix}}} = \frac{1}{m} \sum_{i=1}^{m} \varphi_j(\vec{x}_i), \quad \forall \varphi_j
\]

- Solve linear system

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

with

\[
p_{ij} = \langle \varphi_i(\vec{x}), \varphi_j(\vec{x}) \rangle_{L^2}
\]

\[
b_j = \frac{1}{m} \sum_{i=1}^{m} \varphi_j(\vec{x}_i)
\]
Again: Hierarchical Basis and Data Structures

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

\[
\begin{pmatrix}
\frac{1}{24} & \frac{1}{48} & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{48} & \frac{1}{12} & \frac{1}{48} & 0 & 0 & 0 & 0 \\
0 & \frac{1}{48} & \frac{1}{12} & \frac{1}{48} & 0 & 0 & 0 \\
0 & 0 & \frac{1}{48} & \frac{1}{12} & \frac{1}{48} & 0 & 0 \\
0 & 0 & 0 & \frac{1}{48} & \frac{1}{12} & \frac{1}{48} & 0 \\
0 & 0 & 0 & 0 & \frac{1}{48} & \frac{1}{12} & \frac{1}{24}
\end{pmatrix}
\]

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

- There are optimal complexity algorithms for mat-vec in $O(N)$, 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!
Side-Note: Offline/Online Splitting

\[(P + \lambda I)\bar{\alpha} = \bar{b}\]

- 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 \(\lambda\)
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 \(\lambda\)
- Better alternative: eigendecomposition of \(P\)
  - Offline phase: eigendecomposition \(P = USU^T\)
    - \(U\) orthonormal, \(S\) diagonal
  - Online phase: compute in \(O(N^2)\)
    \[\vec{\alpha} = (P + \lambda I)^{-1}\vec{b} = (USU^T + \lambda UU^T)^{-1}\vec{b} = U(S + \lambda I)^{-1}U^T\vec{b}\]
- 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

<table>
<thead>
<tr>
<th></th>
<th>Tesla P100</th>
<th>FirePro W8100</th>
<th>2xXeon Gold 5120</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>float</td>
<td>double</td>
<td>float</td>
</tr>
<tr>
<td>right-hand side</td>
<td>3323</td>
<td>1827</td>
<td>1581</td>
</tr>
<tr>
<td></td>
<td>36%</td>
<td>39%</td>
<td>38%</td>
</tr>
<tr>
<td>matrix-vector</td>
<td>2450</td>
<td>1503</td>
<td>969</td>
</tr>
<tr>
<td></td>
<td>26%</td>
<td>32%</td>
<td>23%</td>
</tr>
</tbody>
</table>
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
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 $O(m^2)$ 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

```plaintext
for i = 0; i < N; i ← i + 1 do
    for j = 0; j < m; j ← j + 1 do
        for d = 0; d < ds; d ← d + 1
do
    // 6 floating point ops
```

### Algorithm 2: Create graph kernel

```plaintext
for i = 0; i < m; i ← i + 1 do
    for j = 0; j < m; j ← j + 1 do
        for d = 0; d < ds; d ← d + 1
do
    // 4 floating point ops
```

### Algorithm 3: Density MV kernel

```plaintext
for i = 0; i < N; i ← i + 1 do
    for j = 0; j < N; j ← j + 1 do
        for d = 0; d < ds; d ← d + 1
do
    // 14 floating point ops
```

### Algorithm 4: Prune graph kernel

```plaintext
for i = 0; i < m; i ← i + 1 do
    for j = 0; j < N; j ← j + 1 do
        for d = 0; d < ds; d ← d + 1
do
    // 6 · (k + 1) fl. pt ops
```

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

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

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

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

<table>
<thead>
<tr>
<th>Code generator</th>
<th>Generated code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>source &lt;&lt; &quot;__kernel mult(&quot; &lt;&lt; &quot;__global double *p)\n&quot;;</code></td>
<td><code>__kernel mult(__global double *p)</code></td>
</tr>
<tr>
<td><code>if (useLocalMemory) {</code></td>
<td><code>_local double v[5];</code></td>
</tr>
<tr>
<td><code>source &lt;&lt; &quot;__local double v[5];\n&quot;;</code></td>
<td><code>v[0] = p[0];</code></td>
</tr>
<tr>
<td><code>for (int k = 0; k &lt; 5; k += 1) {</code></td>
<td><code>v[1] = p[1];</code></td>
</tr>
<tr>
<td></td>
<td><code>v[3] = p[3];</code></td>
</tr>
</tbody>
</table>
Distributed Approach

- Classical manager-worker scheme

Manager

Worker

Worker

Worker

Worker

OpenCL

OpenCL

OpenCL

OpenCL

Send work package
Distributed Approach

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

Manager

Worker

OpenCL

Worker

OpenCL

Worker

OpenCL

Worker

OpenCL

Receive results
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

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

![Graph showing strong scaling results.](image-url)
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

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
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!

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.

David Pfander, Alexander Heinecke, and Dirk Pflüger.
A new subspace-based algorithm for efficient spatially adaptive sparse grid regression, classification and multi-evaluation.

Data mining on vast data sets as a cluster system benchmark.

Benjamin Peherstorfer, Dirk Pflüger, and Hans-Joachim Bungartz.
Clustering based on density estimation with sparse grids.

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

Dirk Pflüger: *Numerical Data Mining with Sparse Grids at Extreme Scale*
IPAM, Big Data meets Large-Scale Computing, September 25, 2018