

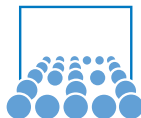
# Algorithms of Scientific Computing

## Hierarchical Methods and Sparse Grids

Tobias Neckel, Dirk Pflüger

Technische Universität München

Summer Term 2010



## Part VII

# Algorithms and Data Structures for Sparse Grids

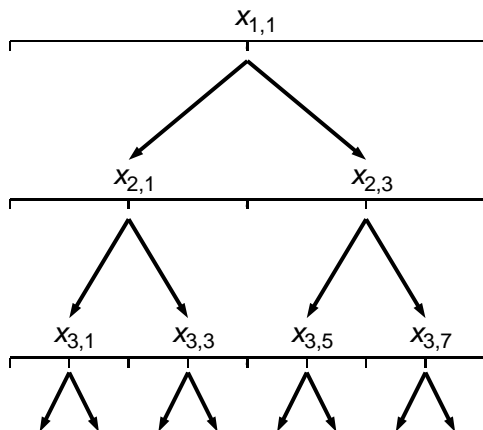
# Algorithms and Data Structures

- We will now look at typical sparse grid algorithms
- Can, e.g., be used for solution of PDE in previous part
- Algorithms depend on data structure:
  - Efficient traversal of sparse grid necessary
  - Thus, we deal with data structures for sparse grids, too

# Data Structures ( $d = 1$ )

- How to store function  $u : [0, 1] \rightarrow \mathbb{R}$  in hierarchical representation (i.e. surplusses  $v_{l,i}$ )?
- Order and store grid points and associated values in binary tree
  - Root is node  $x_{1,1} = 1/2$
  - Children of node  $x_{l,i}$  are – if existent – the grid points  $x_{l+1,2i-1}$  and  $x_{l+1,2i+1}$  of level  $l + 1$
  - Alternative point of view if child does not exist:  
Complete subtree of binary tree starting from child with all surplusses set to 0

# Data Structures ( $d = 1$ ) (2)



# Typical Algorithms ( $d = 1$ )

## Hierarchization and Dehierarchization

- Prototype for typical algorithm (c.f. worksheet 5)
- Our data structure has to allow
  - 1 Iteration over all grid points, considering the hierarchical relations
    - E.g. for hierarchization: first handle all grid points in the support of  $\phi_{l,i}$ , then compute  $v_{l,i}$
  - 2 Access to *hierarchical neighbors*: grid points at interval boundaries of support of  $\phi_{l,i}$  (if possible – exception for points 0 and 1 as not in the tree), e.g. to compute

$$v_{l,i} = u_{l,i} - \frac{1}{2}(u_l + u_r).$$

## Typical Algorithms ( $d = 1$ ) (2)

- Hierarchical neighbors are easy to find geometrically

$$x_{l,i-1}, \quad x_{l,i+1}$$

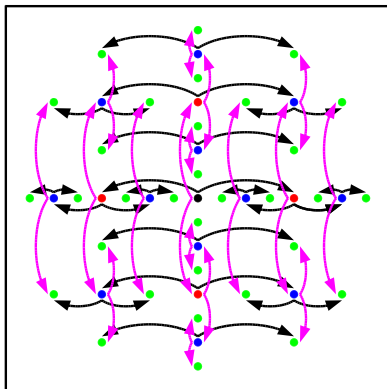
- But have even indices  $\Rightarrow$  really are on another level ( $< l$ )
- In the binary tree structure:
  - Can be found on way from root to node
  - One is parent node
- For hierarchization/dehierarchization: pass hierarchical neighbors as additional parameters
- Developing algorithms:
  - Try to store all information to process one node at the node and its hierarchical neighbors
  - Access to other nodes typically expensive
  - Tree traversal with “supply of hierarchical neighbors” only linear in number of nodes

# Data Structures and Typical Algorithms ( $d > 1$ )

- What data structure to use in more than one dimension?
- Algorithmically: use construction of basis functions as product of one-dimensional hats. Ideally:
  - Use a loop  $1, \dots, d$  over the dimension
  - Apply  $1d$  algorithm on one-dimensional structures in each dimension (see also worksheet 7)
- ⇒ Need access to hierarchical neighbors in each spacial direction; implies to create binary tree structure in each dimension
- Disadvantages:
  - Storage requirements ( $2d$  pointers)
  - High effort to keep structure consistent when inserting or deleting points



# Data Structures and Typical Algorithms ( $d > 1$ ) (2)



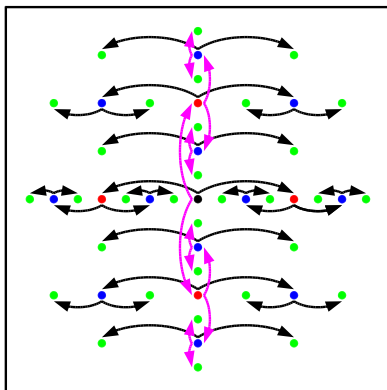
If you could recognize anything, it would be binary tree structures for rows (black) and columns (magenta)

## Data Structures and Typical Algorithms ( $d > 1$ ) (3)

### Oftentimes better:

- Store in a node only two pointers for one direction (e.g.  $x_1$ )
- A binary tree of nodes is a row (a  $1d$  structure parallel to the  $x_1$  axis)
- For next spatial direction  $x_2$ , only a binary tree in  $x_2$  direction required
- Stores one plane parallel to  $x_1 - x_2$  coordinate plane; nodes are the binary trees with  $1d$  structures
- For each additional spatial direction  $x_d$  build binary tree with  $(d - 1)$ -dimensional structures as nodes
- Disadvantage: Access to hierarchical neighbors not that easy any more (except for  $x_1$ -direction)
- But can be achieved without much more computational effort by suitable reordering of loops and tree traversals

# Data Structures and Typical Algorithms ( $d > 1$ ) (4)



Already more clear: One plane (two-dimensional structure) consists of one binary tree (magenta) of which the nodes are binary trees (black) for each row

# Data Structures and Typical Algorithms ( $d > 1$ ) (5)

## Hash table

- Much more comfortable (and not half inefficient) alternative
  - Store magnitudes as target values, with, e.g.,  $(\vec{l}, \vec{i})$  as keys
  - No need to care about tree structures
  - Only have to compute indices of designated node (hierarchical neighbor, ...)
- ⇒ Best solution for your own sparse grid experiments

## Further assumptions on data structures

- Algorithms will assume that all hierarchical neighbors exist for each grid point
- ⇒ If creating grid points adaptively, create them if necessary
- No further assumptions

# Solving Differential Equations on Sparse Grids

## Preliminary Considerations

- Finite elements: method to transform DE to system of linear equations

$$M\vec{v} = \vec{f}$$

for hierarchical surplusses (threaded in vector  $\vec{v}$ )

- Linear system has to be solved
- Problem: matrix in hierarchical basis not sparse, but densely populated
- So many non-zero entries that explicit assembly can be too expensive
- Even worse: prohibits direct solution, e.g. via Cholesky decomposition

## Preliminary Considerations (2)

Less problematic than it seems:

- Matrix can be applied to vector in linear time in length of vector (algorithmically tricky!)
- Use iterative solvers:
  - Only have to implement application of matrix to a given vector

$$\vec{v} \mapsto M\vec{v}$$

(algorithmically challenging and interesting)

- Algorithmically less interesting part is done by someone else (conjugated gradients (CG), ...)
- We consider it at the example of matrix  $B$  ( $L_2$  scalar product)
- Matrix  $A$  for energy scalar product less clear, but can be done similarly

## Matrix-Vector multiplication

Computing  $M\vec{v}$ , with  $M = B$ ,  $b_{i,j} := \int_0^1 \phi_j(x)\phi_i(x) dx$

- Given: hierarchical coefficients  $v_{\vec{l},\vec{i}}$
- Compute in each node  $\vec{l},\vec{i}$  the corresponding component of  $M\vec{v}$ :

$$\int_{\Omega} \phi_{\vec{l},\vec{i}} \left( \sum_{\vec{l}',\vec{j}} v_{\vec{l}',\vec{j}} \phi_{\vec{l}',\vec{j}} \right) d\vec{x} = \sum_{\vec{l}',\vec{j}} \left( \int_{\Omega} \phi_{\vec{l},\vec{i}} \phi_{\vec{l}',\vec{j}} d\vec{x} \right) v_{\vec{l}',\vec{j}} = \sum_{\vec{l}',\vec{j}} \left( \phi_{\vec{l},\vec{i}}, \phi_{\vec{l}',\vec{j}} \right)_2 v_{\vec{l}',\vec{j}}$$

- Think of transport of contributions:
  - Transport surplus at  $\vec{l}',\vec{j}$  with weight  $(\phi_{\vec{l},\vec{i}}, \phi_{\vec{l}',\vec{j}})_2$  to position  $\vec{l},\vec{i}$
  - Everything that arrives at a node is summed up
- It is possible with effort proportional to number of unknowns (squared would be too easy)!

## Multiplication with $B$ , $d = 1$

- Order  $1d$  unknowns by level  $l$  (and within level by index  $i$ , e.g.) – important for mathematics, not for implementation
- Example for  $n = 3$ :

$$B\vec{v} = \left( \begin{array}{ccc|ccc} 1/3 & 1/8 & 1/8 & 1/32 & 3/32 & 3/32 & 1/32 \\ 1/8 & 1/6 & 0 & 1/16 & 1/16 & 0 & 0 \\ 1/8 & 0 & 1/6 & 0 & 0 & 1/16 & 1/16 \\ \hline 1/32 & 1/16 & 0 & 1/12 & 0 & 0 & 0 \\ 3/32 & 1/16 & 0 & 0 & 1/12 & 0 & 0 \\ 3/32 & 0 & 1/16 & 0 & 0 & 1/12 & 0 \\ 1/32 & 0 & 1/16 & 0 & 0 & 0 & 1/12 \end{array} \right) \left( \begin{array}{c} v_{1,1} \\ v_{2,1} \\ v_{2,3} \\ \hline v_{3,1} \\ v_{3,3} \\ v_{3,5} \\ v_{3,7} \end{array} \right)$$

- If follows from hierarchical structure:  
Surplusses have always to be propagated up or down the tree,  
never sideways



# Splitting of $B$

- We now split transports depending on direction in tree structure into
  - procedure *down*, which does transport towards leaves,
  - procedure *up*, which does transport towards root
- Note: splitting not necessary for  $d = 1$ , but helpful to derive operations and necessary for  $d > 1$
- In matrix notation this corresponds to

$$B =: B^D + B^U$$

- $B^D$  corresponds to down, contains entries of  $B$  below diagonal (strictly lower triangular matrix)
- $B^U$  corresponds to up, contains entries on and above diagonal (upper triangular matrix)

## Multiplication with $B$ , $d = 1$ : down

- Down computes in node  $\vec{l}, \vec{i}$

$$\int_{\Omega} \phi_{l,i} \left( \sum_{l' < l, j} v_{l',j} \phi_{l',j} \right) dx,$$

- Value depends only on linear interpolant between hierarchical neighbors of  $x_{\vec{l}, \vec{i}}$ !
- ⇒ Take procedure for dehierarchization
- Modify it to compute integral as

$$h_l \frac{u_l + u_r}{2}.$$

before computing function value  $u_{\vec{l}, \vec{i}}$

## Multiplication with $B$ , $d = 1$ : up

### Up is more difficult...

- Contribution of other basis functions not linear on support

To understand up operations, we consider the following:

- We can neglect the diagonal for the following considerations (unproblematic, as no communication of different nodes)
- $B$  is symmetric, thus  $B^U$  and  $B^D$  are transposed
- Multiplication with  $B^D$  did consist mainly of operations

$$u_{l,i} := u_{l,i} + \frac{u_l + u_r}{2}$$

- They can be described by matrix  $B_{l,i}^D$  (how does it look like?)

## Multiplication with $B$ , $d = 1$ : up (2)

- Multiplication with transposed of matrix  $B_{l,i}^D$  as follows:

$$u_r := u_r + \frac{u_{l,i}}{2}; \quad u_l := u_l + \frac{u_{l,i}}{2}$$

- Multiplication with  $h_l$  corresponds to diagonal matrix (slightly transposed)
  - Now only apply those building blocks in reverse order (due to  $(CD)^T = C^T D^T$ )  $\Rightarrow$  bottom-up tree traversal
- $\Rightarrow$  Multiplication with  $B$  with constant cost per node (even though much more coefficients in  $B$  are non-zero):  $\mathcal{O}(N)$

## Multiplication with $B$ , $d > 1$

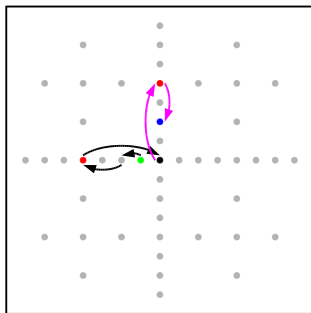
- In multi-dimensional case, we can write weights as products of  $1d$  weights:

$$\left(\phi_{\vec{l}, \vec{j}}, \phi_{\vec{l}', \vec{j}}\right)_2 = \prod_{k=1}^d \left(\phi_{l_k, l'_k}, \phi_{j_k, j_k}\right)_2$$

- This implies general strategy for  $d$ -dimensional problems:
  - Loop over the dimension
  - Loop over all  $1d$  structures in corresponding direction
  - Apply  $1d$  algorithm (up and down) there

# Multiplication with $B$ , $d > 1$ (2)

## Example: transport of contributions

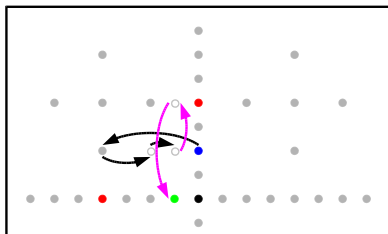


- Transport from grid point  $\vec{i}, \vec{j} = (4, 1), (7, 1)$  to grid point  $\vec{l}, \vec{l} = (1, 3), (1, 3)$
- Up along row (black arrows) computes weight  $(\phi_{1,1}, \phi_{4,7})$ ,  
down along column (magenta) computes weight  $(\phi_{3,3}, \phi_{1,1})$

## Multiplication with $B$ , $d > 1$ (2)

### Troubles with that...

- Nice algorithm, but does not work on sparse grids
- Consider reverse direction – would be like this:



- Three grid points are missing!
- Creating all missing grid points on the fly  $\Rightarrow$  full grid
- It works, if we reorder up and down processes
  - Execute all ups before any down

# All together...

- We have considered suitable data structures  
...and efficient algorithms working on them
- We could now start
  - solving PDEs (iteratively)
  - integrate (and interpolate) multi-dimensional functions
  - and much more...