Mesh Adaptivity and Domain Decomposition for Finite Element Method

Pavel Kůs

Institute of Mathematics, Czech Academy of Sciences, Prague
Max Planck Computing and Data Facility, MPI, München, Germany

In collaboration with Jakub Šístek

Institute of Mathematics, Czech Academy of Sciences, Prague
School of Mathematics, The University of Manchester, UK

PANM 2016
Many ways how to improve performance of finite element codes, two very different strategies are:

- parallel, domain decomposition, the use of supercomputers, very large linear systems
- adaptivity, higher order, the goal is to have smaller linear systems solvable on PC, still with good accuracy

It would be nice to combine both strategies

- Parallel calculations
  - take advantage from similarity of structure over the domain
  - most of the research done in linear solver part
  - Exascale brings many challenges. It could be good alternative to get more from petascale instead.

- Adaptivity
  - different treatment of different areas, based on solution behavior
  - a lot of effort in assembly part, trying to minimize number of DOFs
  - There are certain limits for single PC, no matter how smart the algorithm is.
Many ways how to improve performance of finite element codes, two very different strategies are:

- **Parallel, domain decomposition**, the use of supercomputers, very large linear systems
- **Adaptivity**, higher order, the goal is to have smaller linear systems solvable on PC, still with good accuracy

**It would be nice to combine both strategies**

- **Parallel calculations**
  - take advantage from similarity of structure over the domain
  - most of the research done in linear solver part
  - Exascale brings many challenges. It could be good alternative to get more from petascale instead.

- **Adaptivity**
  - different treatment of different areas, based on solution behavior
  - a lot of effort in assembly part, trying to minimize number of DOFs
  - There are certain limits for single PC, no matter how smart the algorithm is.
Adaptivity and Higher Order Elements

Parallel mesh handler

BDDC method

Parallel implementation

Numerical Results
Adaptivity and Higher Order Elements

Parallel mesh handler

BDDC method

Parallel implementation

Numerical Results
Mesh adaptivity

- Finite element method divides domain to elements
  - On each element polynomial approximation used
  - Finer mesh and higher-order polynomials should lead to better solution
  - It also increases the problem size
  - The goal is to improve the solution where needed

- Common: start with coarse grid and successively improve it based on analysis of the solution obtained in each step

- Different approaches
  - complete re-meshing
  - change of vertices positions, mesh topology is kept ($r$-adaptivity)
  - element refinements ($h$-adaptivity)
  - different polynomial orders ($p$-adaptivity)
  - combination of both ($hp$-adaptivity)

- We need information about the error
  - a-posteriori error estimators
  - exact solution for benchmarks
Mesh adaptivity

- Each approach has its advantages and drawbacks
  - Complete re-meshing leaves the adaptivity to mesh generator, the solver is not aware of it
  - Meshing of complex domains is very expensive
  - $r$-adaptivity keeps topology of the mesh, which is good, but might not be flexible enough
  - For both previous interpolation between completely different meshes
  - $p$-adaptivity not flexible enough
  - $h$-adaptivity does not improve the order of convergence
  - $hp$-adaptivity very challenging for implementation
  - Hanging nodes are issue for both $h$- and $hp$-adaptivity
  - DG-FEM simplifies hanging nodes treatment, but more complicated weak formulation and more expensive
- We will use $h$-adaptivity in the following
Benchmark no. 8 – Schrödinger equation

COMSOL
106496 elements, $p = 2$

Hermes
mesh after $hp$-adapt.

COMSOL
213761 DOFs, error 171 %

Hermes
1908 DOFs, error 0.0089 %
Simplification of the problem

Assumptions on the mesh

- only level-1 hanging nodes allowed
- equal order shape functions, i.e. no $hp$, but higher $p$ fine

We have to keep problem reasonably simple to be able to gain from parallelization

Under such circumstances, dealing with hanging nodes is surprisingly simple and straightforward

We will go through the process in detail now
Hanging nodes

Regular element

- Matrix $G_K$ represents the relationship between local and global DOFs
- Local stiffness matrix distributed to the Global one
- Similarly for right-hand side

\[(L_1, L_2, L_3, L_4) \leftrightarrow (g_2, g_3, g_5, g_6)\]

\[
G_K = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
A_K = \begin{bmatrix}
a(L_1, L_1) & \ldots & a(L_1, L_4) \\
\vdots & \ddots & \vdots \\
a(L_4, L_1) & \ldots & a(L_4, L_4)
\end{bmatrix}
\]

\[
A = \sum_{K \in \mathcal{T}_k} G_K^T A_K G_K
\]
First option – constraints added to global matrix

- There are global degrees of freedom assigned to hanging nodes
- Matrix $A$ assembled as in the regular case
- Solution would be discontinuous $\rightarrow$ the global system has to be extended by constraints $C$
- The matrix $\tilde{A}$ is rectangular, has to be modified before actual solution – various techniques

\[
(L_1, L_2, L_3, L_4) \leftrightarrow (g_2, g_7, g_{11}, g_8)
\]

\[
A = \sum_{K \in T_k} G_K^T A_K G_K
\]

\[
\tilde{A} = \begin{bmatrix} A \\ C \end{bmatrix}
\]

\[
g_{11} = \frac{g_2 + g_5}{2}
\]

\[
C = \left[ 0, \frac{1}{2}, 0, 0, \frac{1}{2}, 0, 0, 0, 0, 0, -1 \right]
\]
Second option – local change of basis

- No global degrees of freedom assigned to hanging nodes
- Matrix $T_K$ used to modify the local stiffness matrix and RHS
- Corresponds to the construction of globally continuous basis function (depicted the one corresponding to $g_5$)
(L_1, L_2, L_3, L_4)^T = T_K (g_2, g_8, g_5, g_9)^T

T_K = \begin{bmatrix}
\frac{1}{2} & 0 & \frac{1}{2} & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}

\tilde{A}_K = T_K^T A_K T_K

A = \sum_{K \in T_k} G_K^T \tilde{A}_K G_K

Second option – local change of basis

- No global degrees of freedom assigned to hanging nodes
- Matrix \( T_K \) used to modify the local stiffness matrix and RHS
- Corresponds to the construction of globally continuous basis function (depicted the one corresponding to \( g_5 \))
Second option – local change of basis

- Works for higher-order basis functions as well
- Works for 2D and 3D for 1-irregular mesh and elements of same order
- Not numbering hanging nodes natural for p4est → we use this approach
Contents

Adaptivity and Higher Order Elements

Parallel mesh handler

BDDC method

Parallel implementation

Numerical Results
Without balancing has no sense in parallel

Most of the refinements would concentrate in those domains, where singularities, boundary or internal layers are present

It might be just few subdomains
Refinements without balancing

Without balancing has no sense in parallel

Most of the refinements would concentrate in those domains, where singularities, boundary or internal layers are present

It might be just few subdomains
Without balancing has no sense in parallel

- Most of the refinements would concentrate in those domains, where singularities, boundary or internal layers are present
- It might be just few subdomains
Without balancing has no sense in parallel

Most of the refinements would concentrate in those domains, where singularities, boundary or internal layers are present

It might be just few subdomains
Refinements without balancing

- Without balancing has no sense in parallel
- Most of the refinements would concentrate in those domains, where singularities, boundary or internal layers are present
- It might be just few subdomains
Without balancing has no sense in parallel

Most of the refinements would concentrate in those domains, where singularities, boundary or internal layers are present

It might be just few subdomains.
Element incidence graph can be created and partitioned

Advantage: usually nice shape of subdomains

Disadvantage: it is not scalable to large number of processors
Partitioning of space-filling curve

- Z-order (space filling) curve can be used
- Each element in the refinement hierarchy might be identified by number, coding bitwise its position and level of refinement
- Can be used both in 2D (quadrilaterals) or 3D (hexahedra)
Partitioning of space-filling curve

- curve will be split equally among individual processors
- Each element in the refinement hierarchy might be identified by number, coding bitwise its position and level of refinement
- Can be used both in 2D (quadrilaterals) or 3D (hexahedra)
Partitioning of space-filling curve

- Can be used for partitioning
- Initial mesh can be small, just to express the geometry. It has to be made of quadrilaterals or hexahedra, which might be limiting.
- Disadvantage: shape of the subdomains far from optimal
Partitioning of space-filling curve

- Can be used for partitioning
- Initial mesh can be small, just to express the geometry. It has to be made of quadrilaterals or hexahedra, which might be limiting.
- Disadvantage: shape of the subdomains far from optimal
Partitioning of space-filling curve

- Can be used for partitioning
- Initial mesh can be small, just to express the geometry. It has to be made of quadrilaterals or hexahedra, which might be limiting.
- Disadvantage: shape of the subdomains far from optimal
Partitioning of space-filling curve

- Can be used for partitioning
- Initial mesh can be small, just to express the geometry. It has to be made of quadrilaterals or hexahedra, which might be limiting.
- Disadvantage: shape of the subdomains far from optimal
Partitioning of space-filling curve

- Can be used for partitioning
- Initial mesh can be small, just to express the geometry. It has to be made of quadrilaterals or hexahedra, which might be limiting.
- Disadvantage: shape of the subdomains far from optimal
Partitioning of space-filling curve

- Can be used for partitioning
- Initial mesh can be small, just to express the geometry. It has to be made of quadrilaterals or hexahedra, which might be limiting.
- Disadvantage: shape of the subdomains far from optimal.
1 Hanging nodes
   - Hanging nodes have to be eliminated
   - They can also appear at the subdomain interface

2 Shape of the subdomains
   - The shape is far from perfect
   - Subdomains might be disconnected or only loosely coupled (e.g. by one node in elasticity)
3 examples of mesh partitioning obtained by p4est

1. regular mesh with number of subdomains leading to aligned subdomains
2. regular mesh with divided into more subdomains leading to unaligned subdomains
3. mesh refined, the same number of subdomains as in 1, but again not aligned subdomains

Even though number of subdomains (in 2) or number of dofs (in 3) changes only slightly from 1, it may have severe impact on BDDC solver performance

Both changes have similar impact
p4est library

- Enables dynamic management of a collection (forest) of adaptive octrees
- p4est is designed to work in parallel and scales to hundreds of thousands of processor cores
- Written in C
- Free software released under GNU General Public License ver. 2
- Used by deal.II and other projects
- Very memory and speed efficient

Work-flow

- Refine/coarsen elements
- Do the mesh re-balancing
- Can enforce the 2:1 balance (1-irregularity rule)
- Provides global numbering of elements and nodes
- Does not provide any finite-element structures
p4est – octants


- p4est can load a general unstructured (but regular – no hanging nodes) mesh as an initial mesh
- Each of the elements is a root of one tree
- Within each tree, refinements can be done
- Only quadrilateral (2D) or hexahedral (3D) elements
An octant $o$ is described by the following

- Its root element – index of the tree within the forest
- Level $l$ – determines its size
- Binary representation of each of its coordinates within the tree

An octant $o$ is described by the following:

- Its root element – index of the tree within the forest
- Level $l$ – determines its size
- Binary representation of each of its coordinates within the tree
An octant $o$ is described by the following:

- Its root element – index of the tree within the forest
- Level $l$ – determines its size
- Binary representation of each of its coordinates within the tree

Coordinates and level are encoded into 1 integer – Morton index

\[ m = (z_0, y_0, x_0, z_1, y_1, x_1, z_2, \ldots, x_{M_l}, l, l, l, l) \]

Ordering of \((t, m)\) gives ordering of elements

Very memory efficient – sufficient to store and communicate contiguous blocks of morton indices

Most of the operations defined in bitwise fashion

Integer has 64 bits – in 2D maximal level \(M_l = 29\), in 3D \(M_l = 19\)

Only active elements (leaves) stored

Contents

Adaptivity and Higher Order Elements

Parallel mesh handler

BDDC method

Parallel implementation

Numerical Results
Domain decomposition – general idea

- Domain is discretized with very small element sizes $h$
- The mesh is divided into subdomains
- Local problems in individual subdomains are solved
- Iterative process needed to enforce continuity
- Coarse problem introduced to make the process scalable
- Important to know the corresponding local DOFs on subdomain interfaces
Domain decomposition – general idea

- Domain is discretized with very small element sizes $h$
- The mesh is divided into subdomains
  - Local problems in individual subdomains are solved
  - Iterative process needed to enforce continuity
  - Coarse problem introduced to make the process scalable
  - Important to know the corresponding local DOFs on subdomain interfaces
Domain decomposition – general idea

- Domain is discretized with very small element sizes $h$
- The mesh is divided into subdomains
- Local problems in individual subdomains are solved
- Iterative process needed to enforce continuity
- Coarse problem introduced to make the process scalable
- Important to know the corresponding local DOFs on subdomain interfaces
Balancing Domain Decomposition based on Constraints

- introduced independently in [Dohrmann (2003)], [Cros (2003)], and [Fragakis and Papadrakakis (2003)] — for Poisson and elasticity problems
- non-overlapping additive DD preconditioner in PCG
- two-level method, additive global coarse correction

Multi-level method

- for many subdomains, exact solution of the global coarse problem may become expensive
- it can then be solved again with the same approach
- we use two- and three-level method
### Balancing Domain Decomposition based on Constraints

- introduced independently in [Dohrmann (2003)], [Cros (2003)], and [Fragakis and Papadrakakis (2003)] — for Poisson and elasticity problems
- non-overlapping additive DD preconditioner in PCG
- two-level method, additive global coarse correction

### Multi-level method

- for many subdomains, exact solution of the global coarse problem may become expensive
- it can than be solved again with the same approach
- we use two- and three-level method
Solving the linear system

An abstract problem

\[ u \in U : a(u, v) = \langle f, v \rangle \quad \forall v \in U \]

- \( a(\cdot, \cdot) \) symmetric positive definite form on \( U \)
- \( \langle \cdot, \cdot \rangle \) is inner product on \( U \)
- \( U \) is finite dimensional space (typically finite element functions)

Matrix form

\[ u \in U : Au = f \]

- \( A \) symmetric positive definite matrix on \( U \)
- \( A \) large, sparse, condition number \( \kappa(A) = \frac{\lambda_{\max}}{\lambda_{\min}} = O(1/h^2) \)
Solving the linear system

An abstract problem

\[ u \in U : a(u, v) = \langle f, v \rangle \quad \forall v \in U \]

- \( a(\cdot, \cdot) \) symmetric positive definite form on \( U \)
- \( \langle \cdot, \cdot \rangle \) is inner product on \( U \)
- \( U \) is finite dimensional space (typically finite element functions)

Matrix form

\[ u \in U : Au = f \]

- \( A \) symmetric positive definite matrix on \( U \)
- \( A \) large, sparse, condition number \( \kappa(A) = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} = \mathcal{O}(1/h^2) \)
Iterative substructuring

- interface $\Gamma$

Reduced (Schur complement) problem on interface $\Gamma$

\[ Su_\Gamma = g \]

- $S$ ... Schur complement matrix
- $g$ ... reduced right-hand side
- solved by PCG
Iterative substructuring

Reduced (Schur complement) problem on interface $\Gamma$

$Su_\Gamma = g$

- $S$ ... Schur complement matrix
- $g$ ... reduced right-hand side
- solved by PCG
Iterative substructuring

Reduced (Schur complement) problem on interface $\Gamma$

$$S u_{\Gamma} = g$$

- $S$ ... Schur complement matrix
- $g$ ... reduced right-hand side
- solved by PCG
The BDDC preconditioner

- $U$ continuous at all nodes at interface
- $\tilde{W}$ continuous at selected coarse dofs
- $W$ no continuity at interface

- continuity at corners, and of averages (arithmetic or weighted) over edges or faces considered
- enough constraints to fix floating subdomains — $a (\cdot, \cdot)$ symmetric positive definite on $\tilde{W}$

Variational form of $M_{BDDC} : r \mapsto u$

$$w \in \tilde{W} : \quad a (w, z) = \langle r, Ez \rangle \quad \forall z \in \tilde{W}$$
$$u = Ez$$
The BDDC preconditioner

\[ U \subset \tilde{W} \subset W \]

- continuous at all nodes at interface
- continuous at selected coarse dofs
- no continuity at interface

- continuity at corners, and of averages (arithmetic or weighted) over edges or faces considered
- enough constraints to fix floating subdomains — \( a(\cdot, \cdot) \) symmetric positive definite on \( \tilde{W} \)

**Variational form of** \( M_{BDDC} : r \mapsto u \)

\[ w \in \tilde{W} : \quad a(w, z) = \langle r, Ez \rangle \quad \forall z \in \tilde{W} \]

\[ u = Ew \]
Disconnected and loosely coupled subdomains

- nullspaces of subdomain matrices unknown a priori
- detect **graph components** of subdomain mesh
- **components independent** during classification of interface into faces, edges and vertices
- corners selected by the face-based algorithm [Šístek et al. (2011)]
- size of local problems unchanged, but larger nullspaces lead locally to more constraints — still potential load imbalance
Disconnected and loosely coupled subdomains

- nullspaces of subdomain matrices unknown a priori
- detect **graph components** of subdomain mesh
- **components independent** during classification of interface into faces, edges and vertices
- corners selected by the face-based algorithm [Šístek et al. (2011)]
- size of local problems unchanged, but larger nullspaces lead locally to more constraints — still potential load imbalance
Disconnected and loosely coupled subdomains

- nullspaces of subdomain matrices unknown a priori
- detect **graph components** of subdomain mesh
- **components independent** during classification of interface into faces, edges and vertices
- corners selected by the face-based algorithm [Šístek et al. (2011)]
- size of local problems unchanged, but larger nullspaces lead locally to more constraints — still potential load imbalance
Adaptivity and Higher Order Elements

Parallel mesh handler

BDDC method

Parallel implementation

Numerical Results
We use **non-overlapping** domain decomposition method, namely **BDDC**, a Balancing Domain Decomposition by Constraints. Experiments with two different libraries:

- **Fempar library**
  - A FORTRAN library for the development of Finite Element Multiphysics PARallel solvers
  - Developed at CIMNE, Barcelona, Spain by the group of Santiago Badia
  - Scales to hundreds of thousands of processor cores
  - Large project, includes all FEM machinery (space discretization, integration, assembling, physic-based preconditioners, ...)
  - Most of the data structures are “flat”, using a lot of arrays, which is not optimal for adaptivity implementation
  - A lot of the code has to be aware of the hanging nodes

- **BDDCML library**
  - A FORTRAN library, BDDC **Multi Level**
  - Developed at IM AS CR by Jakub Šístek
  - Only linear algebra solver
  - Receives discrete system and some geometry information
  - It is much easier to deal with hanging nodes on the interface
We use **non-overlapping** domain decomposition method, namely **BDDC**, a Balancing Domain Decomposition by Constraints. Experiments with two different libraries:

- **Fempar library**
  - A FORTRAN library for the development of Finite Element Multiphysics PARallel solvers
  - Developed at CIMNE, Barcelona, Spain by the group of Santiago Badia
  - Scales to hundreds of thousands of processor cores
  - Large project, includes all FEM machinery (space discretization, integration, assembling, physic-based preconditioners, . . .)
  - Most of the data structures are “flat”, using a lot of arrays, which is not optimal for adaptivity implementation
  - A lot of the code has to be aware of the hanging nodes

- **BDDCML library**
  - A FORTRAN library, BDDC **Multi Level**
  - Developed at IM AS CR by Jakub Šístek
  - Only linear algebra solver
  - Receives discrete system and some geometry information
  - It is much easier to deal with hanging nodes on the interface
Ingredients

1. Experience with adaptivity and higher order finite elements


2. Experience with domain decomposition, BDDCML library


3. Parallel mesh handler p4est

## Parallel implementation

### Parallel FEM solver with AMR

- experimental in-house code
- high order finite elements
- Poisson equation and linear elasticity
- C++ (object oriented) + MPI

### p4est mesh manager for AMR

- rebalancing based on Z-curves
- ANSI C + MPI
- open-source (GPL)
- scalability reported for 1e5–1e6 cores

http://www.p4est.org

### BDDCML equation solver

- Adaptive-Multilevel BDDC
- Fortran 95 + MPI
- open-source (LGPL)
- current version 2.5 (8/6/’15)
- tested on up to 65e3 cores and 2e9 unknowns

http://www.math.cas.cz/~sistek/software/bddcml.html
Discretization is done outside the BDDCML library, which needs

- Subdomain matrices and rhs
- Local to global numbering maps for elements, nodes and DOFs
- Some geometry information
- Various settings and additional information

In each adaptivity step:

1. Let p4est balance mesh (migrate elements) – global
2. Let p4est create global enumeration of Lobatto points – global
3. Create FEM-related data (easy for Poisson equation) – local
4. Build local matrices and rhs, global-local info (using p4est) – local
5. Load all required data to BDDCML – local
6. Let BDDCML solve the problem – global
7. Obtain solution from BDDCML – local
8. Estimate solution error for each element – local
9. Find out what elements have globally large error – global
10. Tell p4est which elements to refine – local
Parallel implementation

- Discretization is done outside the BDDCML library, which needs
  - Subdomain matrices and rhs
  - Local to global numbering maps for elements, nodes and DOFs
  - Some geometry information
  - Various settings and additional information

- In each adaptivity step:
  1. Let p4est balance mesh (migrate elements) – global
  2. Let p4est create global enumeration of Lobatto points – global
  3. Create FEM-related data (easy for Poisson equation) – local
  4. Build local matrices and rhs, global-local info (using p4est),... – local
  5. Load all required data to BDDCML – local
  6. Let BDDCML solve the problem – global
  7. Obtain solution from BDDCML – local
  8. Estimate solution error for each element – local
  9. Find out what elements have globally large error – global
  10. Tell p4est which elements to refine – local
Parallel implementation

- Discretization is done outside the BDDCML library, which needs
  - Subdomain matrices and rhs
  - Local to global numbering maps for elements, nodes and DOFs
  - Some geometry information
  - Various settings and additional information

- In each adaptivity step:
  1. Let p4est balance mesh (migrate elements) – global
  2. Let p4est create global enumeration of Lobatto points – global
  3. Create FEM-related data (easy for Poisson equation) – local
  4. Build local matrices and rhs, global-local info (using p4est), . . . – local
  5. Load all required data to BDDCML – local
  6. Let BDDCML solve the problem – global
  7. Obtain solution from BDDCML – local
  8. Estimate solution error for each element – local
  9. Find out what elements have globally large error – global
  10. Tell p4est which elements to refine – local
Parallel implementation

- Discretization is done outside the BDDCML library, which needs
  - Subdomain matrices and rhs
  - Local to global numbering maps for elements, nodes and DOFs
  - Some geometry information
  - Various settings and additional information

- In each adaptivity step:
  1. Let p4est balance mesh (migrate elements) – global
  2. Let p4est create global enumeration of Lobatto points – global
  3. Create FEM-related data (easy for Poisson equation) – local
  4. Build local matrices and rhs, global-local info (using p4est),… – local
  5. Load all required data to BDDCML – local
  6. Let BDDCML solve the problem – global
  7. Obtain solution from BDDCML – local
  8. Estimate solution error for each element – local
  9. Find out what elements have globally large error – global
  10. Tell p4est which elements to refine – local
Parallel implementation

- Discretization is done outside the BDDCML library, which needs
  - Subdomain matrices and rhs
  - Local to global numbering maps for elements, nodes and DOFs
  - Some geometry information
  - Various settings and additional information

- In each adaptivity step:
  1. Let p4est balance mesh (migrate elements) – global
  2. Let p4est create global enumeration of Lobatto points – global
  3. Create FEM-related data (easy for Poisson equation) – local
  4. Build local matrices and rhs, global-local info (using p4est),… – local
  5. Load all required data to BDDCML – local
  6. Let BDDCML solve the problem – global
  7. Obtain solution from BDDCML – local
  8. Estimate solution error for each element – local
  9. Find out what elements have globally large error – global
  10. Tell p4est which elements to refine – local
Parallel implementation

- Discretization is done outside the BDDCML library, which needs
  - Subdomain matrices and rhs
  - Local to global numbering maps for elements, nodes and DOFs
  - Some geometry information
  - Various settings and additional information

- In each adaptivity step:
  1. Let p4est balance mesh (migrate elements) – global
  2. Let p4est create global enumeration of Lobatto points – global
  3. Create FEM-related data (easy for Poisson equation) – local
  4. Build local matrices and rhs, global-local info (using p4est), . . . – local
  5. Load all required data to BDDCML – local
  6. Let BDDCML solve the problem – global
  7. Obtain solution from BDDCML – local
  8. Estimate solution error for each element – local
  9. Find out what elements have globally large error – global
  10. Tell p4est which elements to refine – local
Parallel implementation

- Discretization is done outside the BDDCML library, which needs
  - Subdomain matrices and rhs
  - Local to global numbering maps for elements, nodes and DOFs
  - Some geometry information
  - Various settings and additional information

- In each adaptivity step:
  1. Let p4est balance mesh (migrate elements) – global
  2. Let p4est create global enumeration of Lobatto points – global
  3. Create FEM-related data (easy for Poisson equation) – local
  4. Build local matrices and rhs, global-local info (using p4est),... – local
  5. Load all required data to BDDCML – local
  6. Let BDDCML solve the problem – global
  7. Obtain solution from BDDCML – local
  8. Estimate solution error for each element – local
  9. Find out what elements have globally large error – global
  10. Tell p4est which elements to refine – local
Which elements to refine

x axis: Estimated element error
y axis: Number of elements

Goal: refine prescribed fraction of elements in each step
It might be too expensive to communicate error estimate of all elements to all processors

The first step is to communicate globally maximal element error
Which elements to refine

2. On each processor, count elements in moderate number of error intervals (the same “bins” on all processors)
3. Communicate to the remaining processors - this can be done

x axis: Estimated element error
y axis: Number of elements
Which elements to refine

4 Using global data, determine error threshold leading to refinement of (approximately) given fraction of elements

5 Refine elements of estimated error larger than threshold

It can be none or all on some processors, but globally as required
Adaptivity and Higher Order Elements

Parallel mesh handler

BDDC method

Parallel implementation

Numerical Results
Numerical Results

- So far we implemented and tested
  - Poisson equation and linear elasticity
  - Adaptivity for arbitrary, but fixed element order
  - Both 2D and 3D (the library is “dimension independent”)
- Performed scalability tests
- Main goal is to investigate the influence of nonstandard shape of partitions on a performance of the DD solver
- Tests performed at Salomon@IT4I
Scaling tests used to verify properties of parallel codes

- If the code does not scale well, we do not get much from using larger computer

- The scaling can be spoiled by different reasons
  - Some data are not distributed (e.g. mesh) → there is a limit of single node RAM memory
  - Too much communication: instead of doing actual work, individual CPUs exchange a lot of data with others
  - Waiting for others (or one “central” CPU) to finish some global job

- Scaling tests types

  - **Strong scaling**: We consider fixed size of the problem. Computational time should decrease proportionally to the increase of number of CPUs.

  - **Weak scaling**: We increase the size of the problem as we increase the number of CPUs, thus the load per CPU remains constant. The computational time should also remain constant.

- There is always a limit (unless “embarrassingly parallel”)

Scaling tests

How to set up the scaling test for adaptivity?

- different trajectories of adaptive computations for changing number of cores
- what time to measure – total solution time, time of the last problem from adaptive loop?
- weak scaling of DD algorithm expected by theory, but setup of weak scaling test for adaptivity unclear
- note that good scaling itself does not necessarily mean good algorithm (it is easier to improve from bad performance for small number of cores)

We used strong scaling tests on the final problem

- refinements are prescribed and always the same, not an adaptive run
- investigate the behavior of DD on these nonstandard meshes
Scaling tests

How to set up the scaling test for adaptivity?

- different trajectories of adaptive computations for changing number of cores
- what time to measure – total solution time, time of the last problem from adaptive loop?
- weak scaling of DD algorithm expected by theory, but setup of weak scaling test for adaptivity unclear
- note that good scaling itself does not necessarily mean good algorithm (it is easier to improve from bad performance for small number of cores)

We used strong scaling tests on the final problem

- refinements are prescribed and always the same, not an adaptive run
- investigate the behavior of DD on these nonstandard meshes
Poisson problem

\[-\Delta u = 1 \text{ on } \Omega, \quad u = 0 \text{ on } \partial \Omega, \quad \Omega = [0, 1]^d, d \in 2, 3.\]

Prescribed refinements

- uniform
- small cube \([0.26, 0.28]^d\)
- sphere with center at the origin
Poisson in 2D and 3D, linear elements

- strong scaling for $120 \cdot 10^6$ dofs in 2D and $60 \cdot 10^6$ dofs in 3D
- computed at Salomon@IT4I
### linear elements, 3D

<table>
<thead>
<tr>
<th>subs.</th>
<th>size</th>
<th>loc. size</th>
<th>$n_\Gamma$</th>
<th>coarse size</th>
<th>PCG its.</th>
</tr>
</thead>
<tbody>
<tr>
<td>32/6</td>
<td>$5.9 \times 10^7$</td>
<td>$1.9 \times 10^6$</td>
<td>$6.6 \times 10^5/280$</td>
<td>423/30</td>
<td>34</td>
</tr>
<tr>
<td>64/8</td>
<td>$5.9 \times 10^7$</td>
<td>$9.3 \times 10^5$</td>
<td>$9.5 \times 10^5/507$</td>
<td>949/63</td>
<td>42</td>
</tr>
<tr>
<td>128/12</td>
<td>$5.9 \times 10^7$</td>
<td>$4.6 \times 10^5$</td>
<td>$1.3 \times 10^6/1624$</td>
<td>2383/187</td>
<td>49</td>
</tr>
<tr>
<td>256/16</td>
<td>$5.9 \times 10^7$</td>
<td>$2.3 \times 10^5$</td>
<td>$1.8 \times 10^6/2514$</td>
<td>5232/256</td>
<td>51</td>
</tr>
<tr>
<td>512/22</td>
<td>$5.9 \times 10^7$</td>
<td>$1.2 \times 10^5$</td>
<td>$2.5 \times 10^6/6110$</td>
<td>$1.1 \times 10^4/492$</td>
<td>58</td>
</tr>
<tr>
<td>1024/32</td>
<td>$5.9 \times 10^7$</td>
<td>$5.8 \times 10^4$</td>
<td>$3.3 \times 10^6/9261$</td>
<td>$2.3 \times 10^4/572$</td>
<td>67</td>
</tr>
<tr>
<td>2048/46</td>
<td>$5.9 \times 10^7$</td>
<td>$2.9 \times 10^4$</td>
<td>$4.4 \times 10^6/1.3 \times 10^4$</td>
<td>$4.5 \times 10^4/748$</td>
<td>71</td>
</tr>
<tr>
<td>4096/64</td>
<td>$5.9 \times 10^7$</td>
<td>$1.5 \times 10^4$</td>
<td>$5.9 \times 10^6/2.0 \times 10^4$</td>
<td>$9.0 \times 10^4/1267$</td>
<td>73</td>
</tr>
</tbody>
</table>

- constant global size
- number of subdomains increases
- size of subdomains decreases
- size of interface and global problem increases
- number of iterations remains reasonable
Poisson in 3D, 2nd and 4th order elements

- strong scaling in 3D for $140 \cdot 10^6$ dofs in using quadratic elements and $20 \cdot 10^6$ dofs using elements of order 4
- computed at *Salomon@IT4I*
Poisson in 3D, strong scaling, 3-level BDDC

4th order elements

<table>
<thead>
<tr>
<th>subs. size</th>
<th>size</th>
<th>loc. size</th>
<th>$n_T$</th>
<th>coarse size</th>
<th>PCG its.</th>
</tr>
</thead>
<tbody>
<tr>
<td>32/6</td>
<td>2.0·10^7</td>
<td>6.2·10^5</td>
<td>5.2·10^5 /308</td>
<td>473/37</td>
<td>24</td>
</tr>
<tr>
<td>64/8</td>
<td>2.0·10^7</td>
<td>3.1·10^5</td>
<td>7.3·10^5 /531</td>
<td>997/53</td>
<td>26</td>
</tr>
<tr>
<td>128/12</td>
<td>2.0·10^7</td>
<td>1.5·10^5</td>
<td>9.6·10^5 /1427</td>
<td>2106/205</td>
<td>31</td>
</tr>
<tr>
<td>256/16</td>
<td>2.0·10^7</td>
<td>7.7·10^4</td>
<td>1.3·10^6 /2556</td>
<td>4690/206</td>
<td>37</td>
</tr>
<tr>
<td>512/22</td>
<td>2.0·10^7</td>
<td>3.9·10^4</td>
<td>1.7·10^6 /5899</td>
<td>9561/437</td>
<td>45</td>
</tr>
<tr>
<td>1024/32</td>
<td>2.0·10^7</td>
<td>1.9·10^4</td>
<td>2.1·10^6 /9459</td>
<td>1.9·10^4 /708</td>
<td>52</td>
</tr>
<tr>
<td>2048/46</td>
<td>2.0·10^7</td>
<td>9658</td>
<td>2.7·10^6 /1.2·10^4</td>
<td>3.9·10^4 /919</td>
<td>52</td>
</tr>
</tbody>
</table>

- constant global size
- number of subdomains increases
- size of subdomains decreases
- size of interface and global problem increases
- number of iterations remains reasonable
Regular and non-regular meshes

- 3 examples of mesh partitioning obtained by p4est
  1. regular mesh with number of subdomains leading to aligned subdomains
  2. regular mesh with divided into more subdomains leading to unaligned subdomains
  3. mesh refined, the same number of subdomains as in 1, but again not aligned subdomains

- Even though number of subdomains (in 2) or number of dofs (in 3) changes only slightly from 1, it may have severe impact on BDDC solver performance
- Both changes have similar impact
## Regular and non-regular meshes

<table>
<thead>
<tr>
<th>NS</th>
<th>ndof</th>
<th>ndof/NS</th>
<th>ninterface</th>
<th>ncoarse</th>
<th>niter</th>
<th>$T_S$</th>
<th>$T_K$</th>
</tr>
</thead>
<tbody>
<tr>
<td>512/22</td>
<td>$1.70 \cdot 10^7$</td>
<td>$3.3 \cdot 10^4$</td>
<td>$1.3 \cdot 10^6/1586$</td>
<td>2863/252</td>
<td>16</td>
<td>3.4</td>
<td>2.1</td>
</tr>
<tr>
<td>513/22</td>
<td>$1.70 \cdot 10^7$</td>
<td>$3.3 \cdot 10^4$</td>
<td>$1.8 \cdot 10^6/3530$</td>
<td>8197/303</td>
<td>22</td>
<td>4.3</td>
<td>3</td>
</tr>
<tr>
<td>512/22</td>
<td>$1.74 \cdot 10^7$</td>
<td>$3.4 \cdot 10^4$</td>
<td>$1.8 \cdot 10^6/5028$</td>
<td>7745/530</td>
<td>30</td>
<td>4.7</td>
<td>3.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NS</th>
<th>num. coarse dofs</th>
<th>time fact. local</th>
<th>time solution local</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>avg</td>
</tr>
<tr>
<td>512/22</td>
<td>6</td>
<td>18</td>
<td>14</td>
</tr>
<tr>
<td>513/22</td>
<td>8</td>
<td>62</td>
<td>39</td>
</tr>
<tr>
<td>512/22</td>
<td>12</td>
<td>63</td>
<td>36</td>
</tr>
</tbody>
</table>
Would Metis be closer to regular mesh?

- Meshes obtained from graph partitioning could have better properties
- Metis (or Parmetis) would not scale to such number of cores as p4est does, but let us compare for moderate number of cores
The same mesh partitioned from graph (Metis) and Z-curve (p4est)

- no significant differences in numbers of iterations
- more tests required in this direction
Internal layer benchmark

$$-\Delta u = f \quad \text{on} \quad (0, 1)^d$$

$$u = \arctan\left(s \cdot (r - \frac{\pi}{3})\right)$$

- solution exhibits sharp internal layer
- $r$ is a distance from a given point
- $s$ controls “steepness” of the layer
Adaptivity in 2D on 8 subdomains

- Adaptivity tested for element orders 1-4 (showed order 1)
- Guided by exact solution, using $H^1$ semi-norm for error calculation
- At the beginning local problems very small, but changing number of subdomains (processors) during the calculation is highly impractical

Iteration 3, mesh and solution
Adaptivity in 2D on 8 subdomains

- Adaptivity tested for element orders 1-4 (showed order 1)
- Guided by exact solution, using $H^1$ semi-norm for error calculation
- At the beginning local problems very small, but changing number of subdomains (processors) during the calculation is highly impractical

Iteration 5, mesh and solution
Adaptivity in 2D on 8 subdomains

- Adaptivity tested for element orders 1-4 (showed order 1)
- Guided by exact solution, using $H^1$ semi-norm for error calculation
- At the beginning local problems very small, but changing number of subdomains (processors) during the calculation is highly impractical

Iteration 8, mesh and solution
Adaptivity in 2D on 8 subdomains

- Adaptivity tested for element orders 1-4 (showed order 1)
- Guided by exact solution, using $H^1$ semi-norm for error calculation
- At the beginning local problems very small, but changing number of subdomains (processors) during the calculation is highly impractical

Iteration 13, mesh and solution
Adaptivity in 2D on 8 subdomains

- Adaptivity tested for element orders 1-4 (showed order 1)
- Guided by exact solution, using $H^1$ semi-norm for error calculation
- At the beginning local problems very small, but changing number of subdomains (processors) during the calculation is highly impractical

Iteration 18, mesh and solution.
Adaptivity in 3D on 8 subdomains

- Adaptivity tested for element orders 1-4 (showed order 1)
- Guided by exact solution, using $H^1$ semi-norm for error calculation
- At the beginning local problems very small, but changing number of subdomains (processors) during the calculation is highly impractical

Iteration 3, mesh and solution
Adaptivity in 3D on 8 subdomains

- Adaptivity tested for element orders 1-4 (showed order 1)
- Guided by exact solution, using $H^1$ semi-norm for error calculation
- At the beginning local problems very small, but changing number of subdomains (processors) during the calculation is highly impractical

Iteration 5, mesh and solution
Adaptivity in 3D on 8 subdomains

- Adaptivity tested for element orders 1-4 (showed order 1)
- Guided by exact solution, using $H^1$ semi-norm for error calculation
- At the beginning local problems very small, but changing number of subdomains (processors) during the calculation is highly impractical

Iteration 8, mesh and solution
Adaptivity in 3D on 2048 subdomains

- Uniform refinements (gray) have expected order for respective polynomial orders
- Adaptivity does not improve convergence order (it is only $h$-adaptivity), but is significantly better
- After several steps error equidistributed
Adaptivity in 3D on 2048 subdomains

Properties of individual steps, 2048 subdomains

<table>
<thead>
<tr>
<th>size</th>
<th>loc. size</th>
<th>coarse size</th>
<th>PCG its.</th>
<th>$T_S$</th>
<th>$T_K$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4913</td>
<td>2</td>
<td>4055/552</td>
<td>9</td>
<td>2.8</td>
<td>1.1</td>
</tr>
<tr>
<td>8594</td>
<td>4</td>
<td>7756/942</td>
<td>29</td>
<td>0.55</td>
<td>1.6</td>
</tr>
<tr>
<td>$2.5 \cdot 10^4$</td>
<td>12</td>
<td>$1.7 \cdot 10^4/1005$</td>
<td>53</td>
<td>0.6</td>
<td>3</td>
</tr>
<tr>
<td>$1.3 \cdot 10^5$</td>
<td>63</td>
<td>$3.6 \cdot 10^4/988$</td>
<td>60</td>
<td>0.67</td>
<td>3.5</td>
</tr>
<tr>
<td>$7.0 \cdot 10^5$</td>
<td>342</td>
<td>$4.5 \cdot 10^4/1109$</td>
<td>54</td>
<td>0.89</td>
<td>3.5</td>
</tr>
<tr>
<td>$3.0 \cdot 10^6$</td>
<td>1445</td>
<td>$4.7 \cdot 10^4/1152$</td>
<td>56</td>
<td>1.6</td>
<td>4.8</td>
</tr>
<tr>
<td>$1.4 \cdot 10^7$</td>
<td>6623</td>
<td>$4.7 \cdot 10^4/1057$</td>
<td>55</td>
<td>2.9</td>
<td>10</td>
</tr>
<tr>
<td>$6.4 \cdot 10^7$</td>
<td>$3.1 \cdot 10^4$</td>
<td>$4.5 \cdot 10^4/1011$</td>
<td>55</td>
<td>10</td>
<td>33</td>
</tr>
<tr>
<td>$2.9 \cdot 10^8$</td>
<td>$1.4 \cdot 10^5$</td>
<td>$4.8 \cdot 10^4/1222$</td>
<td>56</td>
<td>61</td>
<td>130</td>
</tr>
<tr>
<td>$1.3 \cdot 10^9$</td>
<td>$6.3 \cdot 10^5$</td>
<td>$4.7 \cdot 10^4/1088$</td>
<td>51</td>
<td>565</td>
<td>521</td>
</tr>
</tbody>
</table>

- 3-level BDDC, NS = 2048 subdomains, 46 subdomains on 2. level
- tri-linear elements
- Largest calculation for 1.3 billion unknowns
Presented numerical tests

- We presented results for a simple test example on extremely simple geometry.
- The approach should have more general use.
- `p4est` allows to load any unstructured (but regular) initial mesh and then perform all required operations on that.
- There are also variants of FEM, which enforce boundary conditions in a weak sense — rectangular grid is sufficient:
  - no meshing, which is expensive and causes problems for complex geometries.
  - simple mesh handling, fewer element data in memory.
  - simpler moving boundary handling.
Conclusions

Development of parallel FEM with AMR

- level-1 hanging nodes simple to handle and to interface with a DD solver
- trade-off between versatility and complexity
- disconnected and loosely coupled subdomains handled by detecting components of subdomain mesh
- certain load imbalance in subdomains

Future work

- improve linear elasticity
- embedded boundary methods
## Conclusions

### Development of parallel FEM with AMR

- level-1 hanging nodes simple to handle and to interface with a DD solver
- trade-off between versatility and complexity
- disconnected and loosely coupled subdomains handled by detecting components of subdomain mesh
- certain load imbalance in subdomains

### Future work

- improve linear elasticity
- embedded boundary methods
Thank you for your attention.

Pavel Kůš, Jakub Šístek
kus@math.cas.cz, sistek@math.cas.cz
http://users.math.cas.cz/~kus
http://users.math.cas.cz/~sistek

Institute of Mathematics, Czech Academy of Sciences, Prague

BDDCML library webpage
http://users.math.cas.cz/~sistek/software/bddcml.html