

# Primal methods of iterative substructuring

Jakub Šístek

Institute of Mathematics of the AS CR, Prague



*Programs and Algorithms of Numerical Mathematics 16*  
June 5th, 2012



## Presentation based on collaboration with

- **J. Mandel** (University of Colorado Denver) and **B. Sousedík** (University of Southern California) – adaptive BDDC, multilevel BDDC
- **P. Burda, M. Čertíková** and **J. Novotný** (Czech Technical University in Prague) – MPI implementations, selection of corners, engineering problems, flow simulation
- **F. Cirak** (University of Cambridge) – engineering applications to flow problems, BDDC for non-symmetric problems



Some motivation for domain decomposition

Brief overview of domain decomposition methods

Iterative substructuring

Domain decomposition preconditioners

Balancing Domain Decomposition by Constraints (BDDC)

- Algorithm of standard (two-level) BDDC method

- Algorithm of Multilevel BDDC

- Parallel implementation of Multilevel BDDC

- Adaptive selection of constraints

- Parallel implementation of Adaptive BDDC

- Numerical results

Conclusions



## Some motivation for domain decomposition

Brief overview of domain decomposition methods

Iterative substructuring

Domain decomposition preconditioners

## Balancing Domain Decomposition by Constraints (BDDC)

Algorithm of standard (two-level) BDDC method

Algorithm of Multilevel BDDC

Parallel implementation of Multilevel BDDC

Adaptive selection of constraints

Parallel implementation of Adaptive BDDC

Numerical results

## Conclusions



- Using **Finite Element Method (FEM)** for problems described by partial differential equations (PDEs) results in solving very large sparse systems of algebraic equations – difficulties with solution by conventional numerical methods
  - **direct methods** slow due to the problem size – complexity  $\mathcal{O}(1/h^3)$  in 2D,  $\mathcal{O}(1/h^6)$  in 3D,  $h$  – mesh size, state-of-the-art: **multifrontal method**
  - **iterative methods** have attractive complexity  $\mathcal{O}(1/h^n)$ ,  $n$  – space dimension, but become slow due to large condition number – suitable **preconditioner** needed! state-of-the-art: **Krylov subspace methods (PCG, BICGSTAB, GMRES)**
  - **combination of these approaches** – synergy in **domain decomposition (DD) methods**
- way to parallelize FEM – naturally distribute both work and memory requirements
- multigrid (MG) solvers are efficient, but difficult to parallelize and balance load



- Using **Finite Element Method (FEM)** for problems described by partial differential equations (PDEs) results in solving very large sparse systems of algebraic equations – difficulties with solution by conventional numerical methods
  - **direct methods** slow due to the problem size – complexity  $\mathcal{O}(1/h^3)$  in 2D,  $\mathcal{O}(1/h^6)$  in 3D,  $h$  – mesh size, state-of-the-art: **multifrontal method**
  - **iterative methods** have attractive complexity  $\mathcal{O}(1/h^n)$ ,  $n$  – space dimension, but become slow due to large condition number – suitable **preconditioner** needed! state-of-the-art: **Krylov subspace methods (PCG, BICGSTAB, GMRES)**
  - **combination of these approaches** – synergy in **domain decomposition (DD) methods**
- way to parallelize FEM – naturally distribute both work and memory requirements
- multigrid (MG) solvers are efficient, but difficult to parallelize and balance load



- Using **Finite Element Method (FEM)** for problems described by partial differential equations (PDEs) results in solving very large sparse systems of algebraic equations – difficulties with solution by conventional numerical methods
  - **direct methods** slow due to the problem size – complexity  $\mathcal{O}(1/h^3)$  in 2D,  $\mathcal{O}(1/h^6)$  in 3D,  $h$  – mesh size, state-of-the-art: **multifrontal method**
  - **iterative methods** have attractive complexity  $\mathcal{O}(1/h^n)$ ,  $n$  – space dimension, but become slow due to large condition number – suitable **preconditioner** needed! state-of-the-art: **Krylov subspace methods (PCG, BICGSTAB, GMRES)**
  - **combination of these approaches** – synergy in **domain decomposition (DD) methods**
- way to parallelize FEM – naturally distribute both work and memory requirements
- multigrid (MG) solvers are efficient, but difficult to parallelize and balance load



Some motivation for domain decomposition

Brief overview of domain decomposition methods

Iterative substructuring

Domain decomposition preconditioners

Balancing Domain Decomposition by Constraints (BDDC)

- Algorithm of standard (two-level) BDDC method

- Algorithm of Multilevel BDDC

- Parallel implementation of Multilevel BDDC

- Adaptive selection of constraints

- Parallel implementation of Adaptive BDDC

- Numerical results

Conclusions





## Variational setting

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

- $a(\cdot, \cdot)$  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  $\kappa(A) = \frac{\lambda_{\max}}{\lambda_{\min}} = \mathcal{O}(1/h^2)$

Linked together

$$\langle Au, v \rangle = a(u, v) \quad \forall u, v \in U$$



## Variational setting

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

- $a(\cdot, \cdot)$  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  $\kappa(A) = \frac{\lambda_{\max}}{\lambda_{\min}} = \mathcal{O}(1/h^2)$

Linked together

$$\langle Au, v \rangle = a(u, v) \quad \forall u, v \in U$$



## How subdomains are formed?

- overlapping methods – subdomains not disjoint – overlap
- nonoverlapping methods (a.k.a. substructuring) – subdomains disjoint

## How method is used?

- self-standing iterative methods – simple iteration
- one step of the method used as **preconditioner** for another iterative method (PCG, BICGSTAB, GMRES, ...)



## How subdomains are formed?

- overlapping methods – subdomains not disjoint – overlap
- nonoverlapping methods (a.k.a. substructuring) – subdomains disjoint

## How method is used?

- self-standing iterative methods – simple iteration
- one step of the method used as **preconditioner** for another iterative method (PCG, BICGSTAB, GMRES, . . . )



## How subdomain solutions are organised?

- one-by-one – multiplicative methods (better efficiency, worse parallelization)
- simultaneously – **additive methods** (worse efficiency, simpler parallelization)

## How many levels are involved?

- one-level methods – simple, poor performance for large number of subdomains  $N$
- **two-level methods** – involve **coarse level** (good efficiency, worse parallelization) – dominant today
- **multi-level methods** – when coarse problem becomes ‘too large’ (not the same as *multigrid*)



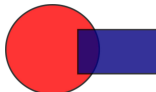
## How subdomain solutions are organised?

- one-by-one – multiplicative methods (better efficiency, worse parallelization)
- simultaneously – **additive methods** (worse efficiency, simpler parallelization)

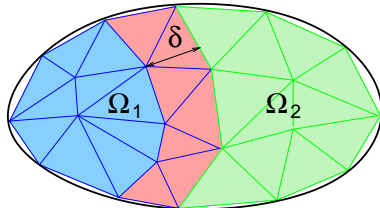
## How many levels are involved?

- one-level methods – simple, poor performance for large number of subdomains  $N$
- **two-level methods** – involve **coarse level** (good efficiency, worse parallelization) – dominant today
- **multi-level methods** – when coarse problem becomes ‘too large’ (not the same as *multigrid*)

- idea goes back to Hermann Schwarz (1870) – for solution of Dirichlet problem on ‘general’ domain – split domain into two simple subdomains with known analytical solution



- nowadays known as **Alternating Schwarz Method**
- multiplicative correction by subdomain problems



- $\Omega_1, \Omega_2 \dots$  subdomains
- $\delta \dots$  size of overlap



- Let us look for a solution of the Poisson's equation given on a domain  $\Omega$  as

$$\begin{aligned}-\Delta u &= f, \\ u &= 0 \quad \text{on } \partial\Omega.\end{aligned}$$

- 1 Given  $u^k$ , solve

$$\begin{aligned}-\Delta u^{k+1/2} &= f \quad \text{on } \Omega_1, \\ u^{k+1/2} &= u^k \quad \text{on } \Gamma_1.\end{aligned}$$

- 2 Given  $u^{k+1/2}$ , solve

$$\begin{aligned}-\Delta u^{k+1} &= f \quad \text{on } \Omega_2, \\ u^{k+1} &= u^{k+1/2} \quad \text{on } \Gamma_2.\end{aligned}$$





- Let us look for a solution of the Poisson's equation given on a domain  $\Omega$  as

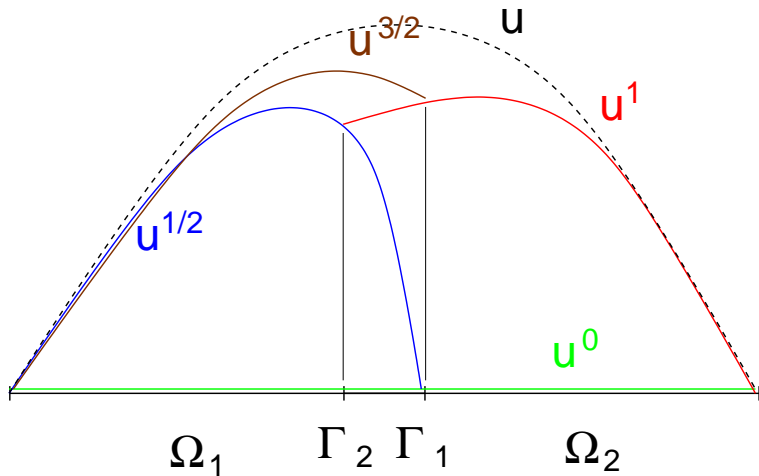
$$\begin{aligned}-\Delta u &= f, \\ u &= 0 \quad \text{on } \partial\Omega.\end{aligned}$$

- 1** Given  $u^k$ , solve

$$\begin{aligned}-\Delta u^{k+1/2} &= f \quad \text{on } \Omega_1, \\ u^{k+1/2} &= u^k \quad \text{on } \Gamma_1.\end{aligned}$$

- 2** Given  $u^{k+1/2}$ , solve

$$\begin{aligned}-\Delta u^{k+1} &= f \quad \text{on } \Omega_2, \\ u^{k+1} &= u^{k+1/2} \quad \text{on } \Gamma_2.\end{aligned}$$





- instead of serial setting, solve problems on both subdomains independently

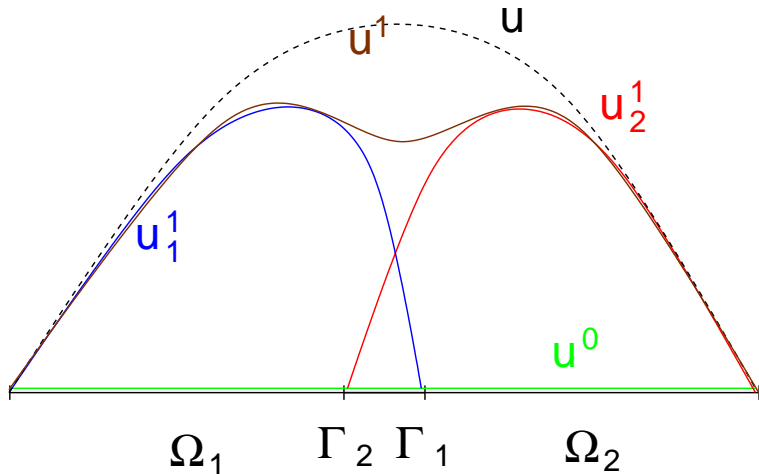
**1** Given  $u^k$ , solve

$$\begin{aligned} -\Delta u_1^{k+1} &= f \text{ on } \Omega_1, \\ u_1^{k+1} &= u^k \text{ on } \Gamma_1, \end{aligned}$$

$$\begin{aligned} -\Delta u_2^{k+1} &= f \text{ on } \Omega_2, \\ u_2^{k+1} &= u^k \text{ on } \Gamma_2. \end{aligned}$$

**2** Get new solution by *addition*

$$u^{k+1} = u_1^{k+1} + u_2^{k+1}$$





## derivatives of Alternating Schwarz Method

- multiplicative correction
- difficult for parallelization
- in general better convergence

## derivatives of Additive Schwarz Method (ASM)

- additive correction
- simple for parallelization
- in general worse convergence
- used as **preconditioners** (e.g. PETSc library – PCASM)
- condition number  $\kappa$  depends on size of overlap  $\delta$  as  $\mathcal{O}(1/\delta)$ , i.e. small overlap – poor preconditioner, large overlap – expensive subdomain solutions
- may involve coarse level  
e.g. [Blaheta et al. (2009)], [Dohrmann, Widlund (2009)]



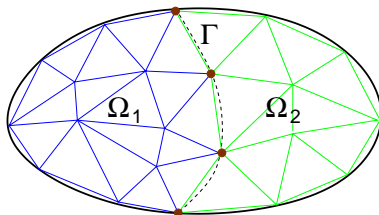
## derivatives of Alternating Schwarz Method

- multiplicative correction
- difficult for parallelization
- in general better convergence

## derivatives of Additive Schwarz Method (ASM)

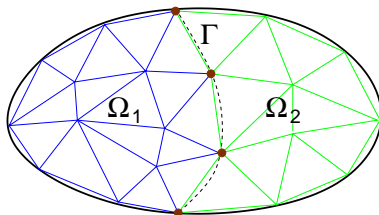
- additive correction
- simple for parallelization
- in general worse convergence
- used as **preconditioners** (e.g. PETSc library – PCASM)
- condition number  $\kappa$  depends on size of overlap  $\delta$  as  $\mathcal{O}(1/\delta)$ , i.e. small overlap – poor preconditioner, large overlap – expensive subdomain solutions
- may involve coarse level  
e.g. [Blaheta et al. (2009)], [Dohrmann, Widlund (2009)]

- idea goes back to **substructuring** – a trick used in seventies to fit larger FE problems into memory



- $\Omega_1, \Omega_2 \dots$  subdomains (substructures)
- $\Gamma \dots$  interface
- unknowns at interface are shared by more subdomains, remaining (interior) unknowns belong to a single subdomain
- the first step is reduction of the problem to the **interface  $\Gamma$**

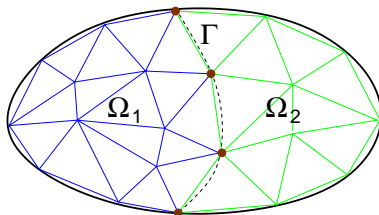
- idea goes back to **substructuring** – a trick used in seventies to fit larger FE problems into memory



- $\Omega_1, \Omega_2 \dots$  subdomains (substructures)
- $\Gamma \dots$  interface
- unknowns at interface are shared by more subdomains, remaining (interior) unknowns belong to a single subdomain
- the first step is reduction of the problem to the **interface  $\Gamma$**



- idea goes back to **substructuring** – a trick used in seventies to fit larger FE problems into memory



- $\Omega_1, \Omega_2 \dots$  subdomains (substructures)
- $\Gamma \dots$  interface
- unknowns at interface are shared by more subdomains, remaining (interior) unknowns belong to a single subdomain
- the first step is reduction of the problem to the **interface  $\Gamma$**



Some motivation for domain decomposition

Brief overview of domain decomposition methods

**Iterative substructuring**

Domain decomposition preconditioners

Balancing Domain Decomposition by Constraints (BDDC)

- Algorithm of standard (two-level) BDDC method

- Algorithm of Multilevel BDDC

- Parallel implementation of Multilevel BDDC

- Adaptive selection of constraints

- Parallel implementation of Adaptive BDDC

- Numerical results

Conclusions



- recall the matrix problem

$$Au = f$$

- reorder unknowns so that those at interior  $u_o^1$  and  $u_o^2$  are first, then interface  $u_\Gamma$

$$\begin{bmatrix} A_{oo}^1 & & A_{o\Gamma}^1 \\ & A_{oo}^2 & A_{o\Gamma}^2 \\ A_{\Gamma o}^1 & A_{\Gamma o}^2 & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_o^1 \\ u_o^2 \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} f_o^1 \\ f_o^2 \\ f_\Gamma \end{bmatrix}$$

- eliminate interior unknowns – subdomain by subdomain = **in parallel**

$$\begin{bmatrix} A_{oo}^1 & & A_{o\Gamma}^1 \\ & A_{oo}^2 & A_{o\Gamma}^2 \\ & & S \end{bmatrix} \begin{bmatrix} u_o^1 \\ u_o^2 \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} f_o^1 \\ f_o^2 \\ g \end{bmatrix}$$

$$S = \sum_{\text{assembly}} A_{\Gamma\Gamma}^i - A_{\Gamma o}^i (A_{oo}^i)^{-1} A_{o\Gamma}^i = \sum_{\text{assembly}} S^i$$

$$g = \sum_{\text{assembly}} f_\Gamma^i - A_{\Gamma o}^i (A_{oo}^i)^{-1} f_o^i = \sum_{\text{assembly}} g^i$$



- recall the matrix problem

$$Au = f$$

- reorder unknowns so that those at interior  $u_o^1$  and  $u_o^2$  are first, then interface  $u_\Gamma$

$$\begin{bmatrix} A_{oo}^1 & & A_{o\Gamma}^1 \\ & A_{oo}^2 & A_{o\Gamma}^2 \\ A_{\Gamma o}^1 & A_{\Gamma o}^2 & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_o^1 \\ u_o^2 \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} f_o^1 \\ f_o^2 \\ f_\Gamma \end{bmatrix}$$

- eliminate interior unknowns – subdomain by subdomain = **in parallel**

$$\begin{bmatrix} A_{oo}^1 & & A_{o\Gamma}^1 \\ & A_{oo}^2 & A_{o\Gamma}^2 \\ & & S \end{bmatrix} \begin{bmatrix} u_o^1 \\ u_o^2 \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} f_o^1 \\ f_o^2 \\ g \end{bmatrix}$$

$$S = \sum_{\text{assembly}} A_{\Gamma\Gamma}^i - A_{\Gamma o}^i (A_{oo}^i)^{-1} A_{o\Gamma}^i = \sum_{\text{assembly}} S^i$$

$$g = \sum_{\text{assembly}} f_\Gamma^i - A_{\Gamma o}^i (A_{oo}^i)^{-1} f_o^i = \sum_{\text{assembly}} g^i$$



- recall the matrix problem

$$Au = f$$

- reorder unknowns so that those at interior  $u_o^1$  and  $u_o^2$  are first, then interface  $u_\Gamma$

$$\begin{bmatrix} A_{oo}^1 & & A_{o\Gamma}^1 \\ & A_{oo}^2 & A_{o\Gamma}^2 \\ A_{\Gamma o}^1 & A_{\Gamma o}^2 & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_o^1 \\ u_o^2 \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} f_o^1 \\ f_o^2 \\ f_\Gamma \end{bmatrix}$$

- eliminate interior unknowns – subdomain by subdomain = **in parallel**

$$\begin{bmatrix} A_{oo}^1 & & A_{o\Gamma}^1 \\ & A_{oo}^2 & A_{o\Gamma}^2 \\ & & S \end{bmatrix} \begin{bmatrix} u_o^1 \\ u_o^2 \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} f_o^1 \\ f_o^2 \\ g \end{bmatrix}$$

$$S = \sum_{assembly} A_{\Gamma\Gamma}^i - A_{\Gamma o}^i (A_{oo}^i)^{-1} A_{o\Gamma}^i = \sum_{assembly} S^i$$

$$g = \sum_{assembly} f_{\Gamma}^i - A_{\Gamma o}^i (A_{oo}^i)^{-1} f_o^i = \sum_{assembly} g^i$$



Reduced (Schur complement) problem

$$Su_{\Gamma} = g$$

- $S$  ... Schur complement with respect to interface
- $g$  ... reduced right hand side
- the process is also known as 'static condensation'

## Algorithm of substructuring

- 1 Form local Schur complements  $S_i$  for each subdomain and build the global Schur complement  $S$  and right-hand side  $g$  (parallel)
- 2 Solve problem for interface unknowns  $Su_{\Gamma} = g$
- 3 Resolve (in parallel) interior unknowns by back-substitution in

$$A_{oo}u_o = f_o - A_{o\Gamma}u_{\Gamma}$$



Reduced (Schur complement) problem

$$Su_{\Gamma} = g$$

- $S$  ... Schur complement with respect to interface
- $g$  ... reduced right hand side
- the process is also known as 'static condensation'

## Algorithm of substructuring

- 1 Form local Schur complements  $S_i$  for each subdomain and build the global Schur complement  $S$  and right-hand side  $g$  (parallel)
- 2 Solve problem for interface unknowns  $Su_{\Gamma} = g$
- 3 Resolve (in parallel) interior unknowns by back-substitution in

$$A_{oo}u_o = f_o - A_{o\Gamma}u_{\Gamma}$$



- if global matrix  $S$  is formed explicitly and problem  $Su_{\Gamma} = g$  solved by a direct method in an efficient (tree-based) way – very much the idea of **multifrontal method** (an MPI version in MUMPS library)
- but the interface problem can be solved by an iterative method (PCG, BICGSTAB, GMRES, ...) – **iterative substructuring**

## Appealing consequences of iterative substructuring

- problem  $Su_{\Gamma} = g$  is usually much better conditioned than the original problem  $Au = f$  – for SPD problems  $\kappa(S) = \mathcal{O}(1/Hh)$  ( $H \gg h$  is the subdomain size) compared to  $\kappa(A) = \mathcal{O}(1/h^2)$  – lower number of iterations
- iterative methods for  $Su_{\Gamma} = g$  are simpler for parallelization
- Krylov subspace methods require only actions of  $S$  on a vector  $p$  – explicit construction of  $S$  may be avoided – large savings of both time and memory – **how?**





- if global matrix  $S$  is formed explicitly and problem  $Su_{\Gamma} = g$  solved by a direct method in an efficient (tree-based) way – very much the idea of **multifrontal method** (an MPI version in MUMPS library)
- but the interface problem can be solved by an iterative method (PCG, BICGSTAB, GMRES, ...) – **iterative substructuring**

## Appealing consequences of iterative substructuring

- problem  $Su_{\Gamma} = g$  is usually much better conditioned than the original problem  $Au = f$  – for SPD problems  $\kappa(S) = \mathcal{O}(1/Hh)$  ( $H \gg h$  is the subdomain size) compared to  $\kappa(A) = \mathcal{O}(1/h^2)$  – lower number of iterations
- iterative methods for  $Su_{\Gamma} = g$  are simpler for parallelization
- Krylov subspace methods require only actions of  $S$  on a vector  $p$  – explicit construction of  $S$  may be avoided – large savings of both time and memory – **how?**



- if global matrix  $S$  is formed explicitly and problem  $Su_{\Gamma} = g$  solved by a direct method in an efficient (tree-based) way – very much the idea of **multifrontal method** (an MPI version in MUMPS library)
- but the interface problem can be solved by an iterative method (PCG, BICGSTAB, GMRES, ...) – **iterative substructuring**

## Appealing consequences of iterative substructuring

- problem  $Su_{\Gamma} = g$  is usually much better conditioned than the original problem  $Au = f$  – for SPD problems  $\kappa(S) = \mathcal{O}(1/Hh)$  ( $H \gg h$  is the subdomain size) compared to  $\kappa(A) = \mathcal{O}(1/h^2)$  – lower number of iterations
- iterative methods for  $Su_{\Gamma} = g$  are simpler for parallelization
- Krylov subspace methods require only actions of  $S$  on a vector  $p$  – explicit construction of  $S$  may be avoided – large savings of both time and memory – **how?**



■ In **setup**:

- 1 factorize matrix  $A_{oo}$  (block diagonal = in parallel)
- 2 form condensed right hand side by solving

$$A_{oo}h = f_o,$$

and inserting  $g = f_r - A_{ro}h$ .

■ In **each iteration**, for given  $p$  construct  $Sp$  as

$$\begin{bmatrix} A_{oo} & A_{or} \\ A_{ro} & A_{rr} \end{bmatrix} \begin{bmatrix} w \\ p \end{bmatrix} = \begin{bmatrix} 0 \\ Sp \end{bmatrix}$$

- 1 Solve (in parallel) *discrete Dirichlet problem*

$$A_{oo}w = -A_{or}p$$

- 2 Get  $Sp$  (in parallel) as

$$Sp = A_{ro}w + A_{rr}p$$

■ **After iterations**, for given  $u_r$ , resolve (in parallel) interior unknowns by back-substitution in

$$A_{oo}u_o = f_o - A_{or}u_r$$



■ In **setup**:

- 1 factorize matrix  $A_{oo}$  (block diagonal = in parallel)
- 2 form condensed right hand side by solving

$$A_{oo}h = f_o,$$

and inserting  $g = f_r - A_{r\Gamma}h$ .

■ In **each iteration**, for given  $p$  construct  $Sp$  as

$$\begin{bmatrix} A_{oo} & A_{o\Gamma} \\ A_{\Gamma o} & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} w \\ p \end{bmatrix} = \begin{bmatrix} 0 \\ Sp \end{bmatrix}$$

- 1 Solve (in parallel) *discrete Dirichlet problem*

$$A_{oo}w = -A_{o\Gamma}p$$

- 2 Get  $Sp$  (in parallel) as

$$Sp = A_{\Gamma o}w + A_{\Gamma\Gamma}p$$

- **After iterations**, for given  $u_\Gamma$ , resolve (in parallel) interior unknowns by back-substitution in

$$A_{oo}u_o = f_o - A_{o\Gamma}u_\Gamma$$



■ In **setup**:

- 1 factorize matrix  $A_{oo}$  (block diagonal = in parallel)
- 2 form condensed right hand side by solving

$$A_{oo}h = f_o,$$

and inserting  $g = f_r - A_{ro}h$ .

■ In **each iteration**, for given  $p$  construct  $Sp$  as

$$\begin{bmatrix} A_{oo} & A_{o\Gamma} \\ A_{\Gamma o} & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} w \\ p \end{bmatrix} = \begin{bmatrix} 0 \\ Sp \end{bmatrix}$$

- 1 Solve (in parallel) *discrete Dirichlet problem*

$$A_{oo}w = -A_{o\Gamma}p$$

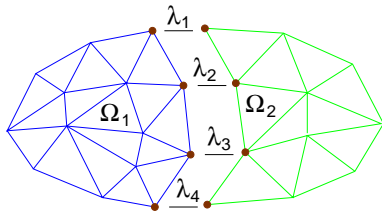
- 2 Get  $Sp$  (in parallel) as

$$Sp = A_{\Gamma o}w + A_{\Gamma\Gamma}p$$

- **After iterations**, for given  $u_\Gamma$ , resolve (in parallel) interior unknowns by back-substitution in

$$A_{oo}u_o = f_o - A_{o\Gamma}u_\Gamma$$

- DD methods based on Schur complement with respect to interface are called **primal DD methods** – they formulate the interface problem in *primal* unknowns
- but another approach exists: disconnect the subdomains completely and enforce continuity at interface weakly by *Lagrange multipliers*  $\lambda$



- this represents adding constraints of the type

$$u_k^1 = u_k^2 \quad \text{or} \quad u_k^1 - u_k^2 = 0$$

where  $u_k$  represents an unknown at the interface



Lead to the saddle point problem with matrix of constraints  $B$

$$\begin{bmatrix} \bar{A} & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \bar{u} \\ \lambda \end{bmatrix} = \begin{bmatrix} \bar{f} \\ 0 \end{bmatrix}$$

- we can eliminate all primal unknowns (**parallel**) and get

$$B\bar{A}^{-1}B^T\lambda = B\bar{A}^{-1}\bar{f}$$

- another interface problem – now for *dual* unknowns – **dual DD methods** (FETI/TFETI, FETI-DP, ...)
- almost the same size as the Schur complement
- solved by an iterative method for  $\lambda$  (same tricks as before)
- when solved for  $\lambda$ , primal unknowns are recovered (in parallel) from  $\bar{A}\bar{u} = \bar{f} - B^T\lambda$
- may involve factorization of singular subdomain matrices –  $\bar{A}^{-1}$  replaced by  $\bar{A}^+$  [Brzobohatý et al. 2011]



Some motivation for domain decomposition

Brief overview of domain decomposition methods

Iterative substructuring

**Domain decomposition preconditioners**

Balancing Domain Decomposition by Constraints (BDDC)

- Algorithm of standard (two-level) BDDC method

- Algorithm of Multilevel BDDC

- Parallel implementation of Multilevel BDDC

- Adaptive selection of constraints

- Parallel implementation of Adaptive BDDC

- Numerical results

Conclusions





Recall the reduced problem

$$Su_r = g$$

that is solved by a Krylov subspace method.

- while  $\kappa(S) = \mathcal{O}(1/Hh)$  is quite nice conditioning, it is not sufficient for large problems – a **preconditioner**  $M \approx S^{-1}$  needed – the best we can do these days is

$$\kappa(MS) \leq C(1 + \log(H/h))^2$$

- convergence independent of number of subdomains and size of the problem, depends only on ratio of subdomain size  $H$  and element size  $h$
- $M$  is realized in parallel by a domain decomposition method



Matrix  $S$  can be obtained by subdomain contributions  $S^i$  as

$$S = \sum_{i=1}^N R^i S^i R^{iT}$$

- $R^i$  ... mapping of subdomain interface unknowns into global interface unknowns
- similar to assembly of global matrix from element matrices

## First idea:

$$S^{-1} \approx \sum_{i=1}^N R^i (S^i)^{-1} R^{iT}$$

- it is still behind the methods but some technical details need to be added



Matrix  $S$  can be obtained by subdomain contributions  $S^i$  as

$$S = \sum_{i=1}^N R^i S^i R^{iT}$$

- $R^i$  ... mapping of subdomain interface unknowns into global interface unknowns
- similar to assembly of global matrix from element matrices

## First idea:

$$S^{-1} \approx \sum_{i=1}^N R^i (S^i)^{-1} R^{iT}$$

- it is still behind the methods but some technical details need to be added



- still neither  $(S^i)^{-1}$  nor  $S^i$  are formed explicitly – instead,  $z_\Gamma^i = (S^i)^{-1} r_\Gamma^i$  is found by solving *discrete Neumann problem*

$$\begin{bmatrix} A_{\text{oo}}^i & A_{\text{o}\Gamma}^i \\ A_{\Gamma\text{o}}^i & A_{\Gamma\Gamma}^i \end{bmatrix} \begin{bmatrix} z_{\text{o}}^i \\ z_\Gamma^i \end{bmatrix} = \begin{bmatrix} 0 \\ r_\Gamma^i \end{bmatrix}$$

- for *floating subdomains*, matrix  $A^i$  is singular – inverse  $(A^i)^{-1}$  has to be replaced by generalized inverse  $(A^i)^+$
- we obtain different solutions for the same interface unknowns from adjacent subdomains – some **averaging** needed to get value of  $z_\Gamma$



- still neither  $(S^i)^{-1}$  nor  $S^i$  are formed explicitly – instead,  $z_\Gamma^i = (S^i)^{-1} r_\Gamma^i$  is found by solving *discrete Neumann problem*

$$\begin{bmatrix} A_{oo}^i & A_{o\Gamma}^i \\ A_{\Gamma o}^i & A_{\Gamma\Gamma}^i \end{bmatrix} \begin{bmatrix} z_o^i \\ z_\Gamma^i \end{bmatrix} = \begin{bmatrix} 0 \\ r_\Gamma^i \end{bmatrix}$$

- for *floating subdomains*, matrix  $A^i$  is singular – inverse  $(A^i)^{-1}$  has to be replaced by generalized inverse  $(A^i)^+$
- we obtain different solutions for the same interface unknowns from adjacent subdomains – some **averaging** needed to get value of  $z_\Gamma$



- still neither  $(S^i)^{-1}$  nor  $S^i$  are formed explicitly – instead,  $z_\Gamma^i = (S^i)^{-1} r_\Gamma^i$  is found by solving *discrete Neumann problem*

$$\begin{bmatrix} A_{oo}^i & A_{o\Gamma}^i \\ A_{\Gamma o}^i & A_{\Gamma\Gamma}^i \end{bmatrix} \begin{bmatrix} z_o^i \\ z_\Gamma^i \end{bmatrix} = \begin{bmatrix} 0 \\ r_\Gamma^i \end{bmatrix}$$

- for *floating subdomains*, matrix  $A^i$  is singular – inverse  $(A^i)^{-1}$  has to be replaced by generalized inverse  $(A^i)^+$
- we obtain different solutions for the same interface unknowns from adjacent subdomains – some **averaging** needed to get value of  $z_\Gamma$



## Neumann–Neumann method

$$S^{-1} \approx \sum_{i=1}^N R^i D^i (S^i)^+ D^i R^{iT}$$

- [De Roeck, Le Tallec (1991)]
- $D^i$  ... weights;  $\sum_{i=1}^N R^i D^i R^{iT} = I$
- one-level method
- convergence deteriorates for growing number of subdomains  $N$  – the approximation of the original problem by preconditioner worsens
- lack of mechanism for propagation of boundary conditions from the physical boundary to subdomains inside
- such mechanism of global correction needed for optimal preconditioning! – **coarse problem**



## BDD

- Balancing Domain Decomposition, [Mandel (1993)]
- adds coarse problem to the Neumann–Neumann method
- two-level method, multiplicative coarse correction
- optimal preconditioning  $\kappa(M_{BDD}S) \leq C(1 + \log(H/h))^2$
- solve consistent singular systems for each subdomain
- coarse problem built upon nullspaces of subdomain matrices

## FETI/TFETI

- Finite Element Tearing and Interconnecting
- [Farhat, Roux (1990)], [Dostál, Horák, Kučera (2006)]
- dual methods
- implicit coarse problem





## BDD

- Balancing Domain Decomposition, [Mandel (1993)]
- adds coarse problem to the Neumann–Neumann method
- two-level method, multiplicative coarse correction
- optimal preconditioning  $\kappa(M_{BDD}S) \leq C(1 + \log(H/h))^2$
- solve consistent singular systems for each subdomain
- coarse problem built upon nullspaces of subdomain matrices

## FETI/TFETI

- Finite Element Tearing and Interconnecting
- [Farhat, Roux (1990)], [Dostál, Horák, Kučera (2006)]
- dual methods
- implicit coarse problem



## FETI-DP

- Finite Element Tearing and Interconnecting - **Dual Primal**
- [Farhat et al. (2000)]
- functions in FETI and BDD have spikes at corners of subdomains – make these unknowns **primal** and add them to the coarse problem – solved as continuous
- important advantage: no floating subdomains – no singular subdomain matrices – can use robust existing sparse direct solvers!



## standard (two-level) BDDC

- Balancing Domain Decomposition based on Constraints
- introduced in [Dohrmann (2003)], convergence theory in [Mandel, Dohrmann (2003)]
- non-overlapping additive DD preconditioner in PCG
- two-level method, additive global coarse correction
- for many subdomains, exact solution of the global coarse problem may become expensive

## extension to multiple levels

- Three-level BDDC [Tu (2007) – 2D, 3D] – basic theory
- Multispace and multilevel BDDC [Mandel, Sousedík, Dohrmann (2008)] - extension to arbitrary number of levels



## standard (two-level) BDDC

- Balancing Domain Decomposition based on Constraints
- introduced in [Dohrmann (2003)], convergence theory in [Mandel, Dohrmann (2003)]
- non-overlapping additive DD preconditioner in PCG
- two-level method, additive global coarse correction
- for many subdomains, exact solution of the global coarse problem may become expensive

## extension to multiple levels

- **Three-level BDDC** [Tu (2007) – 2D, 3D] – basic theory
- **Multispace and multilevel BDDC** [Mandel, Sousedík, Dohrmann (2008)] - extension to arbitrary number of levels



Some motivation for domain decomposition

Brief overview of domain decomposition methods

Iterative substructuring

Domain decomposition preconditioners

## Balancing Domain Decomposition by Constraints (BDDC)

- Algorithm of standard (two-level) BDDC method

- Algorithm of Multilevel BDDC

- Parallel implementation of Multilevel BDDC

- Adaptive selection of constraints

- Parallel implementation of Adaptive BDDC

- Numerical results

Conclusions



Some motivation for domain decomposition

Brief overview of domain decomposition methods

Iterative substructuring

Domain decomposition preconditioners

## Balancing Domain Decomposition by Constraints (BDDC)

- Algorithm of standard (two-level) BDDC method

- Algorithm of Multilevel BDDC

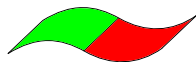
- Parallel implementation of Multilevel BDDC

- Adaptive selection of constraints

- Parallel implementation of Adaptive BDDC

- Numerical results

Conclusions



$U$

continuous  
at all nodes  
at interface

$\subset$



$\widetilde{W}$

continuous  
at selected  
corners

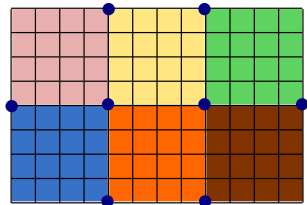
$\subset$



$W$

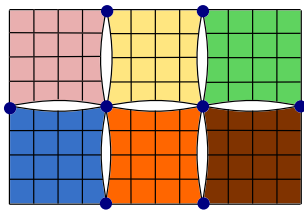
no continuity  
at interface

- enough constraints to fix floating subdomains, i.e. rigid body modes eliminated from the space
- continuity at *corners*, and of averages (arithmetic or weighted) over *edges* or *faces* considered
- $a(\cdot, \cdot)$  symmetric positive definite form on  $\widetilde{W}$
- corresponding matrix  $\widetilde{A}$  symmetric positive definite, almost block diagonal structure, larger dimension than  $A$



*original mesh  
of the problem*

*corresponds to  
space  $U$*



*mesh disconnected  
at interface*

*corresponds to  
space  $\widetilde{W}$*





## Variational form

$$M_{BDDC} : r \mapsto u = Ew, \quad w \in \tilde{W}$$

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

## Matrix form

$$\tilde{A}w = E^T r$$

$$M_{BDDC} r = Ew$$

## Condition number bound [Mandel, Dohrmann (2003)]

$$\kappa = \frac{\lambda_{\max}(M_{BDDC}S)}{\lambda_{\min}(M_{BDDC}S)} \leq \omega = \sup_{w \in \tilde{W}} \frac{\|(I - E)w\|_a^2}{\|w\|_a^2}.$$



## Variational form

$$M_{BDDC} : r \mapsto u = Ew, \quad w \in \tilde{W}$$

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

## Matrix form

$$\tilde{A}w = E^T r$$

$$M_{BDDC} r = Ew$$

## Condition number bound [Mandel, Dohrmann (2003)]

$$\kappa = \frac{\lambda_{\max}(M_{BDDC}S)}{\lambda_{\min}(M_{BDDC}S)} \leq \omega = \sup_{w \in \tilde{W}} \frac{\|(I - E)w\|_a^2}{\|w\|_a^2}.$$



## Variational form

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

## Matrix form

$$\tilde{A}w = E^T r$$
$$M_{BDDC} r = Ew$$

## Condition number bound [Mandel, Dohrmann (2003)]

$$\kappa = \frac{\lambda_{\max}(M_{BDDC}S)}{\lambda_{\min}(M_{BDDC}S)} \leq \omega = \sup_{w \in \tilde{W}} \frac{\|(I - E)w\|_a^2}{\|w\|_a^2}.$$



In implementation, space  $\widetilde{W}$  is decomposed into  $\widetilde{W}_\Delta$  of **independent subdomain spaces** and energy-orthogonal **coarse space**  $\widetilde{W}_\Pi$

$$\widetilde{W} = \widetilde{W}_\Delta \oplus \widetilde{W}_\Pi.$$

## Local energy minimization problems

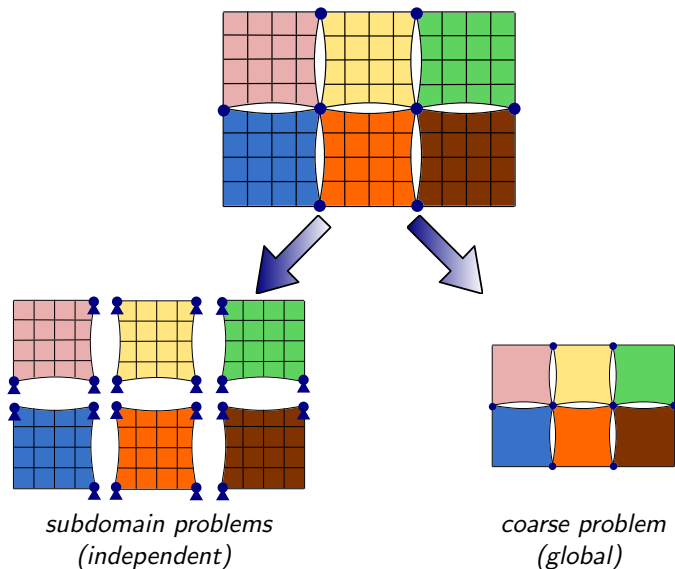
On each subdomain – **coarse degrees of freedom** – basis functions  $\Psi^i$  – prescribed values of coarse degrees of freedom, minimal energy elsewhere,

$$\begin{bmatrix} A^i & C^{iT} \\ C^i & 0 \end{bmatrix} \begin{bmatrix} \Psi^i \\ \Lambda^i \end{bmatrix} = \begin{bmatrix} 0 \\ I \end{bmatrix}.$$

- $A^i$  ... local subdomain stiffness matrix
- $C^i$  ... matrix of constraints – selects unknowns into coarse degrees of freedom

Matrix of **coarse problem**  $A_C$  assembled from local matrices

$$A_{Ci} = \Psi^{iT} A^i \Psi^i = -\Lambda^i.$$



Get residual at interface nodes  $r_\Gamma^{(k)} = g - Su_\Gamma^{(k)}$  and produce **preconditioned residual**  $z_\Gamma^{(k)} = M_{BDDC} r_\Gamma^{(k)}$  by

1. Distribution of residual  
subdomain problems  
for  $i = 1, \dots, N$

$$r^i = E^{iT} r_\Gamma^{(k)}$$

(global) coarse problem

$$r_C = \sum_{i=1}^N R_C^{iT} \Psi^{iT} E^{iT} r_\Gamma^{(k)}$$

2. Correction of solution

$$\begin{bmatrix} A^i & C^{iT} \\ C^i & 0 \end{bmatrix} \begin{bmatrix} z^i \\ \mu^i \end{bmatrix} = \begin{bmatrix} r^i \\ 0 \end{bmatrix}$$

$$A_C u_C = r_C$$

3. Combination of subdomain and coarse corrections

$$z_\Gamma^{(k)} = \sum_{i=1}^N E^i (\Psi^i R_C^i u_C + z^i)$$

**brown formulae** – operations in parallel

Get residual at interface nodes  $r_\Gamma^{(k)} = g - Su_\Gamma^{(k)}$  and produce **preconditioned residual**  $z_\Gamma^{(k)} = M_{BDDC} r_\Gamma^{(k)}$  by

1. Distribution of residual  
subdomain problems  
for  $i = 1, \dots, N$

$$r^i = E^{iT} r_\Gamma^{(k)}$$

(global) coarse problem

$$r_C = \sum_{i=1}^N R_C^{iT} \Psi^{iT} E^{iT} r_\Gamma^{(k)}$$

2. Correction of solution

$$\begin{bmatrix} A^i & C^{iT} \\ C^i & 0 \end{bmatrix} \begin{bmatrix} z^i \\ \mu^i \end{bmatrix} = \begin{bmatrix} r^i \\ 0 \end{bmatrix}$$

$$A_C u_C = r_C$$

3. Combination of subdomain and coarse corrections

$$z_\Gamma^{(k)} = \sum_{i=1}^N E^i (\Psi^i R_C^i u_C + z^i)$$

**brown formulae** – operations in parallel



Get residual at interface nodes  $r_\Gamma^{(k)} = g - Su_\Gamma^{(k)}$  and produce **preconditioned residual**  $z_\Gamma^{(k)} = M_{BDDC} r_\Gamma^{(k)}$  by

1. Distribution of residual  
subdomain problems  
for  $i = 1, \dots, N$

$$r^i = E^{iT} r_\Gamma^{(k)}$$

(global) coarse problem

$$r_C = \sum_{i=1}^N R_C^{iT} \Psi^{iT} E^{iT} r_\Gamma^{(k)}$$

2. Correction of solution

$$\begin{bmatrix} A^i & C^{iT} \\ C^i & 0 \end{bmatrix} \begin{bmatrix} z^i \\ \mu^i \end{bmatrix} = \begin{bmatrix} r^i \\ 0 \end{bmatrix}$$

$$A_C u_C = r_C$$

3. Combination of subdomain and coarse corrections

$$z_\Gamma^{(k)} = \sum_{i=1}^N E^i (\Psi^i R_C^i u_C + z^i)$$

**brown formulae** – operations in parallel





Get residual at interface nodes  $r_\Gamma^{(k)} = g - Su_\Gamma^{(k)}$  and produce **preconditioned residual**  $z_\Gamma^{(k)} = M_{BDDC} r_\Gamma^{(k)}$  by

1. Distribution of residual  
subdomain problems  
for  $i = 1, \dots, N$

$$r^i = E^{iT} r_\Gamma^{(k)}$$

(global) coarse problem

$$r_C = \sum_{i=1}^N R_C^{iT} \Psi^{iT} E^{iT} r_\Gamma^{(k)}$$

2. Correction of solution

$$\begin{bmatrix} A^i & C^{iT} \\ C^i & 0 \end{bmatrix} \begin{bmatrix} z^i \\ \mu^i \end{bmatrix} = \begin{bmatrix} r^i \\ 0 \end{bmatrix}$$

$$A_C u_C = r_C$$

3. Combination of subdomain and coarse corrections

$$z_\Gamma^{(k)} = \sum_{i=1}^N E^i (\Psi^i R_C^i u_C + z^i)$$

**brown formulae** – operations in parallel



Some motivation for domain decomposition

Brief overview of domain decomposition methods

Iterative substructuring

Domain decomposition preconditioners

## Balancing Domain Decomposition by Constraints (BDDC)

Algorithm of standard (two-level) BDDC method

Algorithm of Multilevel BDDC

Parallel implementation of Multilevel BDDC

Adaptive selection of constraints

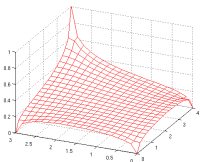
Parallel implementation of Adaptive BDDC

Numerical results

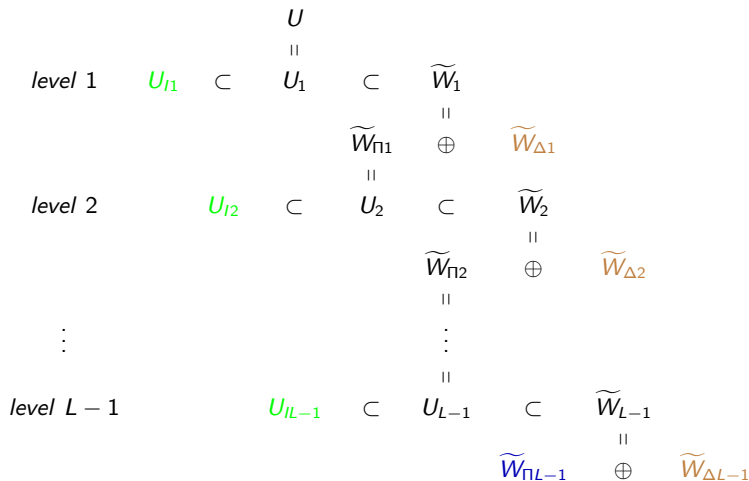
Conclusions



- global coarse problem eventually becomes a bottleneck of parallel processing for very large problems – solve only **approximately** e.g. by
  - several multigrid cycles – [Klawonn, Rheinbach (2010)] (hybrid FETI-DP)
  - by one iteration of BDDC – **Three-level BDDC**
  - recursive application of BDDC – **Multilevel BDDC**
- BDDC is especially suitable for multilevel extension because the coarse problem has the same structure as the original FE problem (unlike in most other DD methods)
- apply BDDC with subdomains playing the role of elements



A basis function from  $\widetilde{W}_\Pi$  is energy minimal subject to given values of coarse degrees of freedom on the substructure. The function is discontinuous across the interfaces between the substructures but the values of coarse degrees of freedom on the different substructures coincide.



Local problems and the coarse problem actually solved are in colour.

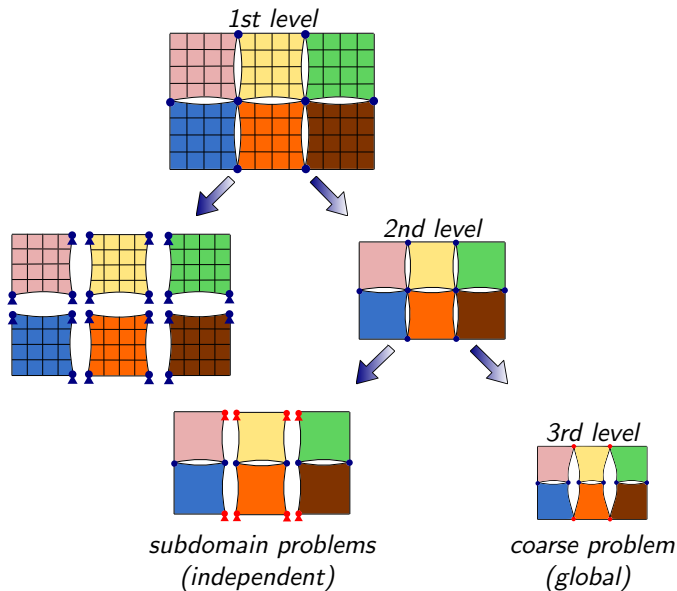


- mathematical efficiency worsens with each additional level

## Theorem [Mandel, Sousedík, Dohrmann (2008)]

The condition number bound  $\kappa(M_{BDDC}S) \leq \omega$  of Multilevel BDDC is given by

$$\kappa(M_{BDDC}S) \leq \omega = \prod_{\ell=1}^{L-1} \omega_{\ell}, \quad \omega_{\ell} = \sup_{w_{\ell} \in \widetilde{W}_{\ell}} \frac{\|(I - E_{\ell}) w_{\ell}\|_a^2}{\|w_{\ell}\|_a^2}.$$





Some motivation for domain decomposition

Brief overview of domain decomposition methods

Iterative substructuring

Domain decomposition preconditioners

## Balancing Domain Decomposition by Constraints (BDDC)

Algorithm of standard (two-level) BDDC method

Algorithm of Multilevel BDDC

Parallel implementation of Multilevel BDDC

Adaptive selection of constraints

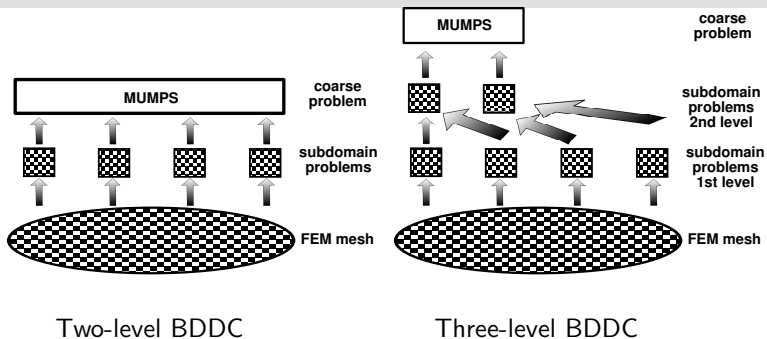
Parallel implementation of Adaptive BDDC

Numerical results

Conclusions

## BDDCML solver library

- <http://www.math.cas.cz/~sistek/software/bddcml.html>
- current version 1.3
- library in Fortran 95 + MPI library
- built on top of MUMPS direct solver (both serial and parallel)
- parallel PCG and BICGSTAB (for overlapping vectors)







Some motivation for domain decomposition

Brief overview of domain decomposition methods

Iterative substructuring

Domain decomposition preconditioners

## Balancing Domain Decomposition by Constraints (BDDC)

Algorithm of standard (two-level) BDDC method

Algorithm of Multilevel BDDC

Parallel implementation of Multilevel BDDC

**Adaptive selection of constraints**

Parallel implementation of Adaptive BDDC

Numerical results

Conclusions



## Theorem (Mandel, Dohrmann, Tezaur '05)

*The abstract BDDC preconditioner satisfies*

$$\kappa = \frac{\lambda_{\max}(M_{BDDC}S)}{\lambda_{\min}(M_{BDDC}S)} \leq \omega = \sup_{w \in \widetilde{W}} \frac{\|(I - E)w\|_a^2}{\|w\|_a^2}.$$

- $\|\cdot\|_a^2 = a(\cdot, \cdot) = \langle S\cdot, \cdot \rangle$  ... energy norm on space  $\widetilde{W}$
- $S$  – Schur complement with respect to interface
- $E$  is the operator of projection

$$E : \widetilde{W} \rightarrow U, \quad \text{Range}(E) = U$$



## Generalized eigenvalue problem

The operator norm  $\omega$  corresponds to the largest eigenvalue of

$$(I - E)^T S (I - E) w = \lambda S w.$$

Select  $k$  largest eigenvalues and corresponding eigenvectors  $\lambda_i, w_i, i = 1, \dots, k$  and define a row of a matrix  $D$  for each of them as

$$d_i = w_i^T (I - E)^T S (I - E),$$

$$D = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \end{bmatrix}.$$



Redefine space

$$\widetilde{W} = \left\{ w \in \widetilde{W}^{init} : Dw = 0 \right\}.$$

Functions from  $\widetilde{W}$  are  $\left\{ (I - E)^T S (I - E) \right\}$ -orthogonal to  $k$  eigenvectors that spoil the condition number the most.

## Theorem (Mandel, Sousedík, Šístek (online in 2011))

*The abstract BDDC preconditioner based on  $\widetilde{W}$  satisfies*

$$\kappa = \frac{\lambda_{\max}(M_{BDDC}S)}{\lambda_{\min}(M_{BDDC}S)} \leq \omega_D = \lambda_{k+1}.$$

A simple consequence of Courant-Fisher-Weyl minimax principle.



Because the solution of the global generalized eigenvalue problem would be prohibitively expensive, solve a number of small eigenproblems **localized to pairs of adjacent subdomains**.

Subdomains  $\Omega_i$  and  $\Omega_j$  are called *adjacent*, if they share a face. They form a *pair* denoted by  $ij$  and  $\mathcal{A}$  denotes the set of all such pairs.

## Local eigenproblem for $ij$ -pair

$$(I - E_{ij})^T S_{ij} (I - E_{ij}) w_{ij} = \lambda_{ij} S_{ij} w_{ij}$$

## Definition

Heuristic *indicator of the condition number*  $\tilde{\omega}$  is defined as

$$\tilde{\omega} = \max_{ij \in \mathcal{A}} \lambda_{ij}^{max}.$$

**Positive side effect** – recognizes troublesome faces  $\Rightarrow$  do not add constraints where they are not necessary, i.e. ‘adaptivity’.



## Algorithm

To generate constraints that will guarantee that the condition number indicator  $\tilde{\omega} \leq \tau$  for a given target value  $\tau$ :

Consider arithmetic averages on all edges as initial constraints.

**For all faces**  $\mathcal{F}_{ij}$

**1** Compute the largest local eigenvalues and corresponding eigenvectors, until the first  $m^{ij}$  is found such that  $\lambda_{m^{ij}}^{ij} \leq \tau$ , put  $k = 1, \dots, m^{ij} - 1$ .

**2** Compute the constraint weights  $d_k^{ij} = \begin{bmatrix} d_k^i & d_k^j \end{bmatrix}$  as

$$d_k^{ij} = w_k^{ijT} (I - E^{ij})^T S_{ij} (I - E^{ij}).$$

**3** Take one block, e.g.,  $d_k^i$  and keep nonzero weights for the face  $\mathcal{F}_{ij}$ .

**4** Add this row to local matrices of constraints  $C_i$  and  $C_j$ .



Some motivation for domain decomposition

Brief overview of domain decomposition methods

Iterative substructuring

Domain decomposition preconditioners

## Balancing Domain Decomposition by Constraints (BDDC)

- Algorithm of standard (two-level) BDDC method

- Algorithm of Multilevel BDDC

- Parallel implementation of Multilevel BDDC

- Adaptive selection of constraints

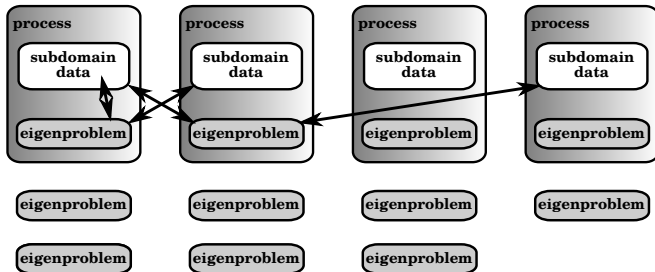
- Parallel implementation of Adaptive BDDC**

- Numerical results

Conclusions



- independent distribution of subdomains and their pairs may lead to difficult communication pattern
- solve eigenproblems in rounds with size of number of processes
- within each round, first determine communication pattern, then perform LOBPCG iterations



An example of a possible parallel layout of local eigenproblems with communication pattern marked for two of them.





Some motivation for domain decomposition

Brief overview of domain decomposition methods

Iterative substructuring

Domain decomposition preconditioners

## Balancing Domain Decomposition by Constraints (BDDC)

Algorithm of standard (two-level) BDDC method

Algorithm of Multilevel BDDC

Parallel implementation of Multilevel BDDC

Adaptive selection of constraints

Parallel implementation of Adaptive BDDC

**Numerical results**

Conclusions



## IBM SP6

Location: CINECA, Italy

Architecture: IBM P6-575 Infiniband Cluster

Processor Type: IBM Power6, 4.7 GHz

Computing Cores: 5376

Computing Nodes: 168

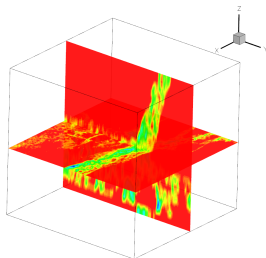
RAM: 21 TB (128 GB/node)

access gained through the *HPC Europa 2* project

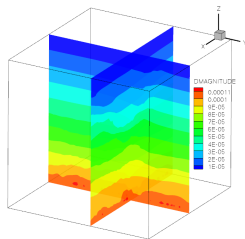


graphics from CINECA website

- problem of geocomposite provided by Prof. Blaheta and Dr. Starý (Institute of Geonics of AS CR)
- cubic sample, edge 75 mm
- 11.8M linear tetrahedral elements, 6.1M unknowns
- arithmetic averages on edges and faces
- required precision ... relative residual =  $\frac{\|res\|}{\|g\|} < 10^{-6}$



*material distribution*



*displacement*



number of procs	64	128	256	512	1024
<b>2 levels</b> (1024/1), <b>46</b> PCG its, cond. $\sim 50$					
set-up phase (sec)	61.0	37.7	25.7	23.2	39.5
iterations (sec)	22.3	19.9	27.8	44.9	97.5
<b>3 levels</b> (1024/128/1), <b>56</b> PCG its, cond. $\sim 79$					
set-up phase (sec)	49.5	29.0	18.4	12.6	11.0
iterations (sec)	28.5	22.6	16.7	14.7	13.2
<b>4 levels</b> (1024/128/16/1), <b>131</b> PCG its, cond. $\sim 568$					
set-up phase (sec)	49.4	28.6	17.8	12.3	9.1
iterations (sec)	60.6	33.2	21.2	15.4	11.8



## Darwin

Location: University of Cambridge, UK

Architecture: Dell PowerEdge 1950 1U Cluster

Processor Type: Intel Xeon 5100 (Woodcrest), 2 cores, 3.0 GHz

Computing Cores: 2048

Computing Nodes: 512 (2 CPUs each)

RAM: 4096 GB (2 GB/core )



graphics from website of High Performance Computing Services of University of Cambridge



- full Taylor–Hood finite elements
- comparison of PETSc (built-in ASM + BiCGstab) and BDDCML (BDDC + BiCGstab)
- partitions by METIS
- size of subdomain problems  $\sim 50k$

			PETSc		BDDC (2-l)		BDDC (3-l)		
elms	size	cores	its	time	its	time	subs/lev	its.	time
$40^3$	1.7M	32	282	533s	18	122s	32/4/1	22	126s
$50^3$	3.2M	64	396	805s	19	132s	64/8/1	25	205s
$64^3$	6.7M	128	384	536s	21	186s	128/16/1	30	194s
$80^3$	13.1M	256	n/a	n/a	21	178s	256/32/1	36	201s
$100^3$	25.4M	512	n/a	n/a	20	205s	512/64/1	35	211s

## Fox

Location: CTU Supercomputing Centre, Prague

Architecture: SGI Altix UV

Processor Type: Intel Xeon 2.67GHz

Computing Cores: 72

Computing Nodes: 12

RAM: 576 GB (8 GB/core)

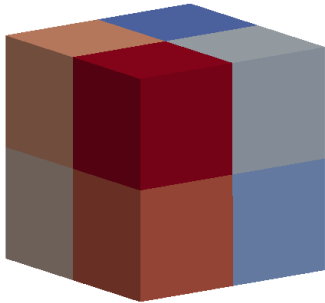


Image courtesy of SGI

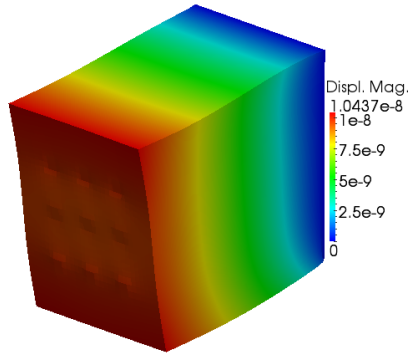


- nonlinear elasticity with St. Venant–Kirchhoff constitutive law
- domain of unit cube with nine stiff rods with Young modulus  $E_2$ , remaining material of variable  $E_1$ , with  $E_2/E_1$  called *contrast*
- loaded by own weight
- $32^3$ ,  $64^3$ , and  $128^3$  tri-linear cubic elements,  $2^3$  and  $4^3$  subdomains, four tested cases:
  - 1 8 subdomains,  $H/h = 16$ ,
  - 2 8 subdomains,  $H/h = 32$ ,
  - 3 64 subdomains,  $H/h = 16$ ,
  - 4 64 subdomains,  $H/h = 32$ .
- required precision . . . relative residual =  $\frac{\|res\|}{\|g\|} < 10^{-6}$
- number of computed eigenpairs set to 10 or 15
- maximum number of LOBPCG iterations limited by 15
- tolerance on residual in LOBPCG set to  $10^{-9}$
- LOBPCG preconditioned by local BDDC [Sousedík 2010]

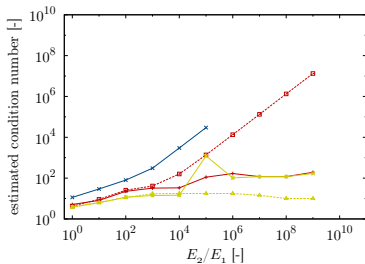
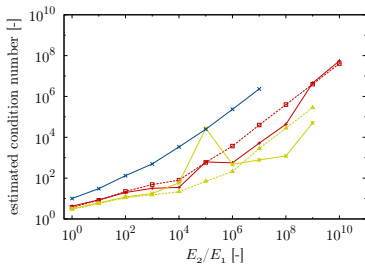
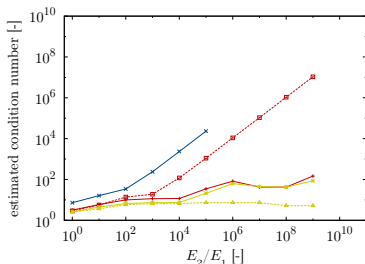
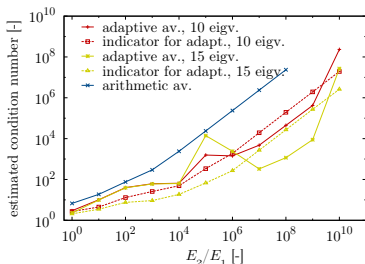




Example of division.

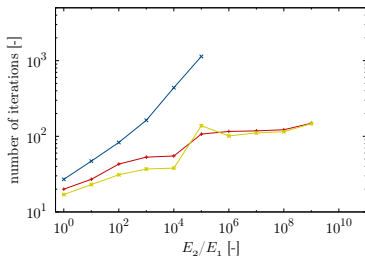
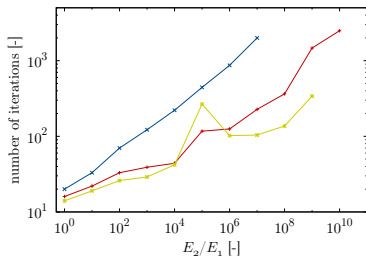
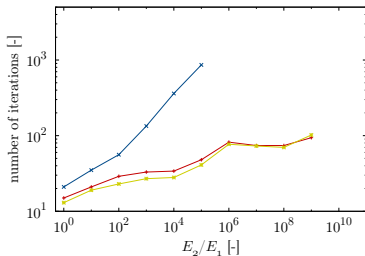
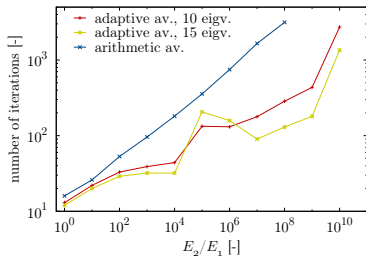


Deformed shape (magnified)  
coloured by magnitude of  
displacement.

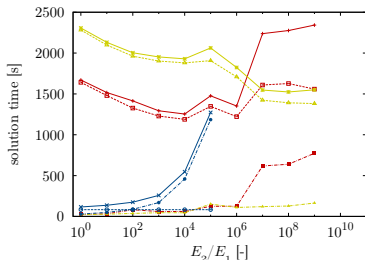
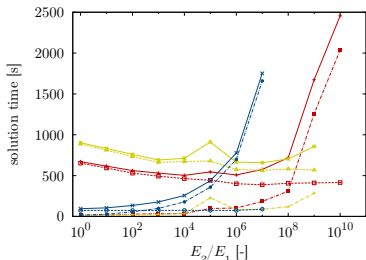
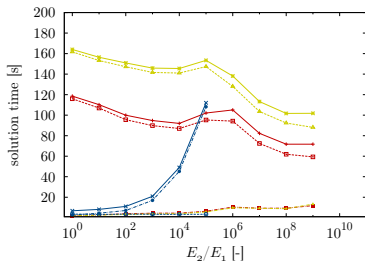
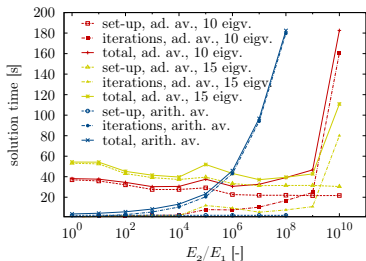


$32^3$  elems,  $H/h = 16$  (top left),  $64^3$  elems,  $H/h = 16$  (top right),  
 $64^3$  elems,  $H/h = 32$  (bot. left),  $128^3$  elems,  $H/h = 32$  (bot. right)

# Number of iterations vs. contrast



$32^3$  elems,  $H/h = 16$  (top left),  $64^3$  elems,  $H/h = 16$  (top right),  
 $64^3$  elems,  $H/h = 32$  (bot. left),  $128^3$  elems,  $H/h = 32$  (bot. right)



$2^3$  elems,  $H/h = 16$  (top left),  $64^3$  elems,  $H/h = 16$  (top right),  
 $64^3$  elems,  $H/h = 32$  (bot. left),  $128^3$  elems,  $H/h = 32$  (bot. right)



Some motivation for domain decomposition

Brief overview of domain decomposition methods

Iterative substructuring

Domain decomposition preconditioners

Balancing Domain Decomposition by Constraints (BDDC)

- Algorithm of standard (two-level) BDDC method

- Algorithm of Multilevel BDDC

- Parallel implementation of Multilevel BDDC

- Adaptive selection of constraints

- Parallel implementation of Adaptive BDDC

- Numerical results

Conclusions



## Implementation of Multilevel BDDC

- powerful combination of Krylov subspace methods with preconditioners by combination of iterative substructuring and domain decomposition
- multilevel extension useful for maintaining scalability on large number of subdomains despite the worse mathematical efficiency
- adaptive BDDC able to solve problems not solvable by standard BDDC, but expensive for simple problems

## Future work

- parallel implementation of **Adaptive Multilevel BDDC** – may keep the efficiency for multilevel BDDC (Matlab tests in [Sousedík (2010)])
- apply and optimize BDDCML for flow problems – CPU hours provided by PRACE-DECI project HIFLY (11/2011 – 10/2012)







## Implementation of Multilevel BDDC

- powerful combination of Krylov subspace methods with preconditioners by combination of iterative substructuring and domain decomposition
- multilevel extension useful for maintaining scalability on large number of subdomains despite the worse mathematical efficiency
- adaptive BDDC able to solve problems not solvable by standard BDDC, but expensive for simple problems

## Future work

- parallel implementation of **Adaptive Multilevel BDDC** – may keep the efficiency for multilevel BDDC (Matlab tests in [Šousedík (2010)])
- apply and optimize BDDCML for flow problems – CPU hours provided by PRACE-DECI project HIFLY (11/2011 – 10/2012)



-  SMITH, B. F., BJØRSTAD, P. E., AND GROPP, W. D.  
*Domain decomposition.*  
Cambridge University Press, Cambridge, 1996.
-  QUARTERONI, A., AND VALLI, A.  
*Domain decomposition methods for partial differential equations.*  
Oxford University Press, New York, 1999.
-  TOSELLI, A., AND WIDLUND, O.  
*Domain decomposition methods—algorithms and theory.*  
Springer-Verlag, Berlin, 2005.
-  KRUIS, J.  
*Domain decomposition methods for distributed computing.*  
Saxe-Coburg Publications, Stirling, 2006.

And chapters on DD are present in many other FEM as well as LA books.