Multilevel Balancing Domain Decomposition at Extreme Scales

Alberto F. Martín, Santiago Badia, Javier Principe

LSSC Team, Centre Internacional de Mètodes Numèrics a l’Enginyeria (CIMNE)
Castelldefels, Spain

Universitat Politècnica de Catalunya
Barcelona, Spain

International Conference on Preconditioning Techniques for Scientific and Industrial Applications
Eindhoven, June 17-19, 2015
1. Introduction and motivation

2. BDDC preconditioner

3. (Exact/Inexact) BDDC MPI-parallel implementation

4. MultiLevel BDDC MPI-parallel implementation

5. Conclusions and future work
1. Introduction and motivation

2. BDDC preconditioner

3. (Exact/Inexact) BDDC MPI-parallel implementation

4. MultiLevel BDDC MPI-parallel implementation

5. Conclusions and future work
Problem statement

- **Variational problem**: Find
  \[ u \in V_h : \quad a(u, v) = (f, v), \quad \text{for any } v \in V_h, \]
  assuming \( a(\cdot, \cdot) \) symmetric, coercive (e.g. Laplacian or linear elasticity)

  \[ \rightarrow \text{Build a conforming FE space } V_h \subset H^1_0(\Omega) \text{ (also for dG)} \]

- **Algebraic problem**: Find
  \[ x \in \mathbb{R}^n : \quad Ax = b, \]
  \( A \) is a large, sparse, and symmetric positive definite (also for nonsym.)
Problem statement

• **Variational problem:** Find

\[ u \in V_h : \quad a(u, v) = (f, v), \quad \text{for any } v \in V_h, \]

assuming \( a(\cdot, \cdot) \) symmetric, coercive (e.g. Laplacian or linear elasticity)

→ Build a conforming FE space \( V_h \subset H^1_0(\Omega) \) (also for dG)

• **Algebraic problem:** Find

\[ x \in \mathbb{R}^n : \quad Ax = b, \]

\( A \) is a large, sparse, and symmetric positive definite (also for nonsym.)
Problem statement

- **Variational problem**: Find
  \[ u \in V_h : \quad a(u, v) = (f, v), \quad \text{for any } v \in V_h, \]
  assuming \( a(\cdot, \cdot) \) symmetric, coercive (e.g. Laplacian or linear elasticity)
  \[ \rightarrow \text{Build a conforming FE space } V_h \subset H^1_0(\Omega) \text{ (also for dG)} \]

- **Algebraic problem**: Find
  \[ x \in \mathbb{R}^n : \quad Ax = b, \]
  \( A \) is a large, sparse, and symmetric positive definite (also for nonsym.)

**Motivation:**
Efficient exploitation of distributed-memory machines for large scale FE problems \( \Rightarrow \)
**Domain decomposition framework**
- interior DoFs \( (I) \); \( \bullet \): interface DOFs \( (\Gamma) \)
Numerical solution of $Ax = b$

- Solve iteratively $M^{-1}Ax = M^{-1}b \rightarrow \tilde{A} = \tilde{b}$
- Preconditioner $M$ aims at **accelerating convergence**
- $M$ should be a parallel, cheap-to-invert, “good” approximation of $A$
Numerical solution of $Ax = b$

- Solve iteratively $M^{-1}Ax = M^{-1}b \rightarrow \tilde{A} = \tilde{b}$
- Preconditioner $M$ aims at accelerating convergence
- $M$ should be a parallel, cheap-to-invert, “good” approximation of $A$
Numerical solution of $Ax = b$

- Solve iteratively $M^{-1}Ax = M^{-1}b \rightarrow \tilde{A} = \tilde{b}$
- Preconditioner $M$ aims at accelerating convergence
- $M$ should be a parallel, cheap-to-invert, “good” approximation of $A$

Solve $Ax = b$ by $M$-PCG
Set-up $M$
call PCG($A,M,b,x^0$)
Numerical solution of $Ax = b$

- Solve iteratively $M^{-1}Ax = M^{-1}b \rightarrow \tilde{A} = \tilde{b}$
- Preconditioner $M$ aims at accelerating convergence
- $M$ should be a parallel, cheap-to-invert, “good” approximation of $A$

Solve $Ax = b$ by $M$-PCG
- Set-up $M$
- call PCG($A,M,b,x^0$)

PCG (In: $(A, M, f, x^0)$, Out: $x$)

\[
\begin{align*}
  r^0 &:= f - Ax^0 \\
  z^0 &:= M^{-1}r^0 \\
  p^0 &:= z^0 \\
  \text{for } j = 0, \ldots, \text{till convergence do} \\
  s^{j+1} &= Ap^j \\
  \alpha^j &:= (r^j, z^j)/(s^{j+1}, p^j) \\
  x^{j+1} &= x^j + \alpha^j p^j \\
  r^{j+1} &= r^j - \alpha^j s^j \\
  z^{j+1} &= M^{-1}r^{j+1} \\
  \beta^j &:= (r^{j+1}, z^{j+1})/(r^j, z^j) \\
  p^{j+1} &= z^{j+1} + \beta^j p^j \\
\end{align*}
\]

end for
Numerical solution of $Ax = b$

- Solve iteratively $M^{-1}Ax = M^{-1}b \rightarrow \tilde{A} = \tilde{b}$
- Preconditioner $M$ aims at accelerating convergence
- $M$ should be a parallel, cheap-to-invert, “good” approximation of $A$

Solve $Ax = b$ by $M$-PCG

Set-up $M$

call PCG($A, M, b, x^0$)

PCG (In: $(A, M, f, x^0)$, Out: $x$)

$$r^0 := f - Ax^0$$
$$z^0 := M^{-1}r^0$$
$$p^0 := z^0$$

for $j = 0, \ldots$, till convergence do

$$s^{j+1} := Ap^j$$
$$\alpha^j := (r^j, z^j)/(s^{j+1}, p^j)$$
$$x^{j+1} := x^j + \alpha^j p^j$$
$$r^{j+1} := r^j - \alpha^j s^j$$
$$z^{j+1} := M^{-1}r^{j+1}$$
$$\beta^j := (r^{j+1}, z^{j+1})/(r^j, z^j)$$
$$p^{j+1} := z^{j+1} + \beta^j p^j$$

end for

Weak scaling for 2D Laplacian with $H/h=256$

Weak scaling: $\uparrow P$, fixed $N/P$
Outline

1. Introduction and motivation
2. BDDC preconditioner
3. (Exact/Inexact) BDDC MPI-parallel implementation
4. MultiLevel BDDC MPI-parallel implementation
5. Conclusions and future work
BDDC Balancing DD by constraints

BDDC preconditioner [Dohrmann, 2003]

- Replace $V_h$ by $\tilde{V}_h$ (reduced continuity)
- Define the injection $I : \tilde{V}_h \rightarrow V_h$ (weight, comm and add)
- Find $\tilde{x}_h \in \tilde{V}_h$ such that:

$$a(\tilde{x}_h, \tilde{v}_h) = \langle I^t r_h, \tilde{v}_h \rangle, \quad \forall \tilde{v}_h \in \tilde{V}_h$$

and obtain $z_h = M_{BDDC} r_h = \mathcal{E} I \tilde{x}_h$

- Last correction: $\mathcal{E}$ is the harmonic extension of the boundary values, which implies local Dirichlet solvers
BDDC preconditions [Dohrmann, 2003]

- Replace $V_h$ by $\tilde{V}_h$ (reduced continuity)
- Define the injection $I : \tilde{V}_h \rightarrow V_h$
  (weight, comm and add)
- Find $\tilde{x}_h \in \tilde{V}_h$ such that:
  \[
a(\tilde{x}_h, \tilde{v}_h) = \langle I^t r_h, \tilde{v}_h \rangle, \quad \forall \tilde{v}_h \in \tilde{V}_h
  \]
  and obtain $z_h = M_{BDDC} r_h = \mathcal{E} I \tilde{x}_h$
- Last correction: $\mathcal{E}$ is the harmonic extension of the boundary values, which implies local Dirichlet solvers
BDDC preconitioner [Dohrmann, 2003]

- Replace $V_h$ by $\tilde{V}_h$ (reduced continuity)
- Define the injection $I : \tilde{V}_h \to V_h$ (weight, comm and add)
- Find $\tilde{x}_h \in \tilde{V}_h$ such that:
  
  $$a(\tilde{x}_h, \tilde{v}_h) = \langle I^t r_h, \tilde{v}_h \rangle, \quad \forall \tilde{v}_h \in \tilde{V}_h$$

  and obtain $z_h = M_{BDDC} r_h = \mathcal{E} I \tilde{x}_h$

- Last correction: $\mathcal{E}$ is the harmonic extension of the boundary values, which implies local Dirichlet solvers
BDDC preconditioner [Dohrmann, 2003]

- Alternatively, find $\tilde{x} \in \mathbb{R}^\tilde{n}$ such that:
  \[ \tilde{A}\tilde{x} = l^t r \]
  and obtain
  \[ z = M_{BDDC} r = E l \tilde{x} \]
- $\tilde{A}$ is a sub-assembled global matrix (only assembled the red corners in the figure)
- $E = \begin{bmatrix} 0 & -A_{II}^{-1}A_{I\Gamma} \\ 0 & l_{\Gamma} \end{bmatrix}$,
  with $A_{II} = diag\left( A_{II}^{(1)}, A_{II}^{(2)}, \ldots, A_{II}^{(P)} \right)$
BDDC preconditioning

- Let $\tilde{V}_h = [\tilde{v}_o \quad \tilde{v}_e]$ and decompose $\tilde{V}_h$ as

$$\tilde{V}_h = \tilde{V}_F \oplus \tilde{V}_C,$$

with

$$\tilde{V}_F = [\tilde{v}_o \quad 0], \quad \tilde{V}_C \perp \tilde{A} \quad \tilde{V}_F$$

- Now, problem split into fine-grid ($\tilde{x}_F$) and coarse-grid ($\tilde{x}_C$) correction
BDDC preconditioning

- Let $\tilde{V}_h = [\tilde{v}_0 \tilde{v}_c]$ and decompose $\tilde{V}_h$ as

$$\tilde{V}_h = \tilde{V}_F \oplus \tilde{V}_C,$$

with

$$\begin{align*}
\tilde{V}_F &= [\tilde{v}_0 0] \\
\tilde{V}_C &\perp \tilde{A} \tilde{V}_F
\end{align*}$$

- Now, problem split into fine-grid ($\tilde{x}_F$) and coarse-grid ($\tilde{x}_C$) correction
BDDC preconditioning

- Let $\tilde{V}_h = [\tilde{v}_o \ \tilde{v}_c]$ and decompose $\tilde{V}_h$ as
  
  $$\tilde{V}_h = \tilde{V}_F \oplus \tilde{V}_C\text{, with } \begin{cases} 
  \tilde{V}_F = [\tilde{v}_o \ 0] \\
  \tilde{V}_C \perp \tilde{A} \tilde{V}_F
  \end{cases}$$

- Now, problem split into fine-grid ($\tilde{x}_F$) and coarse-grid ($\tilde{x}_C$) correction

Fine-grid correction ($\tilde{x}_F$)

- Find $\tilde{x}_F \in \mathbb{R}^{\tilde{n}}$ such that
  
  $$\tilde{A}\tilde{x}_F = I^t r, \text{ constrained to } (\tilde{x}_F)_o = 0$$

- Equivalent to $P$ independent problems

  Find $\tilde{x}^{(i)}_F \in \mathbb{R}^{\tilde{n}^{(i)}}$ such that

  $$A^{(i)}\tilde{x}^{(i)}_F = I^t_i r, \text{ constrained to } (\tilde{x}^{(i)}_F)_o = 0$$

- $\tilde{V}_h$
BDDC preconditioning

- Let $\tilde{V}_h = [\tilde{v}_o \ \tilde{v}_c]$ and decompose $\tilde{V}_h$ as

$$\tilde{V}_h = \tilde{V}_F \oplus \tilde{V}_C,$$

with

$$\tilde{V}_F = [\tilde{v}_o \ 0]$$

$$\tilde{V}_C \perp \tilde{A} \tilde{V}_F$$

- Now, problem split into fine-grid ($\tilde{x}_F$) and coarse-grid ($\tilde{x}_C$) correction

Coarse-grid correction ($\tilde{x}_C$)

Computation of $\tilde{V}_C = \text{span}\{\Phi_1, \Phi_2, \ldots, \Phi_{n_c}\}$

- Find $\Phi \in \mathbb{R}^{\tilde{n} \times n_c}$ such that

$$\tilde{A}\tilde{\Phi} = 0, \text{ constrained to } \Phi_\circ = I$$

- Equivalent to $P$ independent problems

Find $\Phi^{(i)} \in \mathbb{R}^{\tilde{n} \times n_c^{(i)}}$ such that

$$A^{(i)}\Phi^{(i)} = 0, \text{ constrained to } \Phi^{(i)}_\circ = I$$
BDDC coarse corner function

Circle domain partitioned into 9 subdomains

$\Phi_j (\tilde{V}_C$’s basis vector)
BDDC preconditioning

- Let $\tilde{V}_h = [\tilde{v}_o \tilde{v}_c]$ and decompose $\tilde{V}_h$ as

$$\tilde{V}_h = \tilde{V}_F \oplus \tilde{V}_C,$$

with

$$\begin{cases} 
\tilde{V}_F = [\tilde{v}_o \ 0] \\
\tilde{V}_C \perp \tilde{A} \tilde{V}_F 
\end{cases}$$

- Now, problem split into fine-grid ($\tilde{x}_F$) and coarse-grid ($\tilde{x}_C$) correction

**Coarse-grid correction ($\tilde{x}_C$)**

Assembly and solution of coarse-grid problem

$$A_C = \text{assembly}(\Phi^t A^{(i)} \Phi), \quad \text{Solve } A_C \alpha_c = \Phi^t l^t r, \quad \tilde{x}_C = \Phi \alpha_c$$

Coarse-grid problem is

- **Global**, i.e. couples all subdomains
- But much **smaller** than original problem (size $n_C$)
- Potential **loss of parallel efficiency with $P$**
Coarse dof’s definition

**Key aspect:** Selection of coarse dofs, i.e. continuity among subdomains

- Weak scalability ($\kappa(M_{\text{BDDC}A})$ constant for fixed $N/P$ and $\uparrow P$)
- $N/P$ large in practice $\sim O(10^{4-5})$
- BDDC(ce) and BDDC(cef) require much less iterations in 3D
- But at the expense of a more costly coarse-grid problem

<table>
<thead>
<tr>
<th>Coarse dofs vs. $\kappa(M_{\text{BDDC}A})$:</th>
<th>$d = 2$</th>
<th>$d = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuity on corners</td>
<td>[1 + d^{-2}\log^2\left(\frac{N}{P}\right)]</td>
<td>(\frac{N}{P})[1 + d^{-2}\log^2\left(\frac{N}{P}\right)]</td>
</tr>
<tr>
<td>Continuity of mean value on edges too</td>
<td>[1 + d^{-2}\log^2\left(\frac{N}{P}\right)]</td>
<td>[1 + d^{-2}\log^2\left(\frac{N}{P}\right)]</td>
</tr>
<tr>
<td>Continuity of mean value on faces too</td>
<td>-</td>
<td>[1 + d^{-2}\log^2\left(\frac{N}{P}\right)]</td>
</tr>
</tbody>
</table>
BDDC coarse edge function

Circle domain partitioned into 9 subdomains

\( \Phi_j (\tilde{V}_C \text{'s basis vector}) \)
1. Introduction and motivation
2. BDDC preconditioner
3. (Exact/Inexact) BDDC MPI-parallel implementation
4. MultiLevel BDDC MPI-parallel implementation
5. Conclusions and future work
Why BDDC for extreme scales?

1. (Mathematically supported) **extremely aggressive coarsening**
   
   \(10^5 \text{ -- } 10^6\) size reduction between fine/coarse level

2. The coarse matrix has a similar **sparsity** as the original matrix

3. Coarse/local components can be computed **in parallel** (like additive)

4. ALL local + coarse problems can be solved **inexactley** (AMG-cycle)

5. A **multilevel** extension is possible (for extreme core counts)
Why BDDC for extreme scales?

1. (Mathematically supported) **extremely aggressive coarsening**
   
   
   (10^5 – 10^6 size reduction between fine/coarse level)

2. The coarse matrix has a similar **sparsity** as the original matrix

3. Coarse/local components can be computed **in parallel** (like additive)

4. ALL local + coarse problems can be solved **inexactely** (AMG-cycle)

5. A **multilevel** extension is possible (for extreme core counts)
Why BDDC for extreme scales?

1. (Mathematically supported) **exremely aggressive coarsening**
   
   \(10^5 - 10^6\) size reduction between fine/coarse level

2. The coarse matrix has a similar **sparsity** as the original matrix

3. Coarse/local components can be computed **in parallel** (like additive)

4. All local + coarse problems can be solved **inexactl**y (AMG-cycle)

5. A **multilevel** extension is possible (for extreme core counts)
Why BDDC for extreme scales?

1. (Mathematically supported) extremely aggressive coarsening (\(10^5 - 10^6\) size reduction between fine/coarse level)

2. The coarse matrix has a similar sparsity as the original matrix

3. Coarse/local components can be computed in parallel (like additive)

4. ALL local + coarse problems can be solved inexactely (AMG-cycle)

5. A multilevel extension is possible (for extreme core counts)
Why BDDC for extreme scales?

1. (Mathematically supported) **extremely aggressive coarsening**
   
   (10^5 – 10^6 size reduction between fine/coarse level)

2. The coarse matrix has a similar **sparsity** as the original matrix

3. Coarse/local components can be computed **in parallel** (like additive)

4. **ALL local + coarse** problems can be solved **inexactly** (AMG-cycle)

5. A **multilevel** extension is possible (for extreme core counts)
Why BDDC for extreme scales?

1. (Mathematically supported) **extremely aggressive coarsening**
   
   \((10^5 - 10^6\) size reduction between fine/coarse level)

2. The coarse matrix has a similar **sparsity** as the original matrix

3. Coarse/local components can be computed **in parallel** (like additive)

4. ALL local + coarse problems can be solved **inexactly** (AMG-cycle)

5. A **multilevel** extension is possible (for extreme core counts)

- (1)-(2) always exploited in BDDC implementations
- Let us see how to exploit (3), in order to reduce **synchronization** and boost scalability (**overlapped** implementation)
Naive parallel implementation

- All MPI tasks have f-g duties and one/several have also c-g duties
- Computation of f-g/c-g duties serialized (but they are independent!)
- \( T_C \propto O(P^2) \rightarrow \text{idling} \approx PT_C \)
- \( \text{mem} \propto O(P^{\frac{4}{3}}) \rightarrow \text{mem per core rapidly exceeded} \)
Naive parallel implementation

- All MPI tasks have f-g duties and one/several have also c-g duties
- Computation of f-g/c-g duties serialized (but they are independent!)
- \( T_C \propto O(P^2) \rightarrow \text{idling} \simeq PT_C \)
- mem \( \propto O(P^{4/3}) \rightarrow \text{mem per core rapidly exceeded} \)

Overlapped implementation

- Our approach

- MPI tasks have either f-g OR c-g duties
- f-g/c-g corrections OVERLAPPED in time
- c-g duties can be MASKED with f-g ones duties
- OpenMP/MPI-based (MPI later on this talk) solutions possible for c-g correction
Overlapping regions

Solve $Ax = b$ by BDDC-PCG

Set-up $M_{\text{BDDC}}$

$r^0 := b - Ax^0$

$x^0 := x^0 + R_i A^{-1}_{ii} R_i^t r^0$

call PCG$(A, M_{\text{BDDC}}, b, x^0)$

PCG

$r^0 := b - Ax^0$

$z^0 := M_{\text{BDDC}}^{-1} r^0$

$p^0 := z^0$

for $j = 0, \ldots$, till CONV do

$s^{j+1} = Ap^j$

\ldots

$z^{j+1} := M_{\text{BDDC}}^{-1} r^{j+1}$

\ldots

end for
Overlapping regions

Solve $Ax = b$ by BDDC-PCG

Set-up $M_{\text{BDDC}}$

$r^0 := b - Ax^0$

$x^0 := x^0 + R_i A_i^{-1} R_i^t r^0$

call PCG($A, M_{\text{BDDC}}, b, x^0$)

PCG

$r^0 := b - Ax^0$

$z^0 := M_{\text{BDDC}}^{-1} r^0$

$p^0 := z^0$

for $j = 0, \ldots, \text{till CONV}$ do

$s^{j+1} = A p^j$

\ldots

$z^{j+1} := M_{\text{BDDC}}^{-1} r^{j+1}$

\ldots

end for

<table>
<thead>
<tr>
<th>Fine-grid tasks</th>
<th>Coarse-grid task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify local coarse DoFs</td>
<td>Gather coarse-grid DoFs</td>
</tr>
<tr>
<td>Symb fact($G_{A_F(i)}$) $O(n_i^3)$</td>
<td>Symb fact($G_{A_C}$) $O(P_i^3)$</td>
</tr>
<tr>
<td>Symb fact($G_{A_{II}(i)}$) $O(n_i^4)$</td>
<td></td>
</tr>
<tr>
<td>Num fact($A_F(i)$) $O(n_i^2)$</td>
<td></td>
</tr>
<tr>
<td>Compute $\Phi_i$ $O(n_i^3)$</td>
<td></td>
</tr>
<tr>
<td>$A_C(i) := \Phi_i A(i) \Phi_i$</td>
<td></td>
</tr>
<tr>
<td>Gather $A_C(i)$</td>
<td></td>
</tr>
<tr>
<td>Num fact($A_{II}(i)$) $O(n_i^3)$</td>
<td>$A_C := \text{assemble}(A_C(i))$</td>
</tr>
<tr>
<td>$x_0 := x_0 + R_i A_{II(i)}^{-1} R_i^t r_0$ $O(n_i^3)$</td>
<td>Num fact($A_C$) $O(P^2)$</td>
</tr>
<tr>
<td>$r_0 := b - Ax_0$</td>
<td></td>
</tr>
<tr>
<td>$r(i) := l_i^t r$</td>
<td></td>
</tr>
<tr>
<td>$r_C(i) := \Phi_i^t r(i)$</td>
<td></td>
</tr>
<tr>
<td>Gather $r_C(i)$</td>
<td></td>
</tr>
<tr>
<td>Compute $s_F(i)$ $O(n_i^4)$</td>
<td>$r_C := \text{assemble}(r_C(i))$</td>
</tr>
<tr>
<td>Solve $A_C z_C = r_C$ $O(P_i^3)$</td>
<td></td>
</tr>
<tr>
<td>Scatter $z_C$ into $z_C(i)$</td>
<td></td>
</tr>
<tr>
<td>$s_C(i) := \Phi_i z_C(i)$</td>
<td></td>
</tr>
<tr>
<td>$z(i) := l_i(s_F(i) + s_C(i))$</td>
<td></td>
</tr>
<tr>
<td>$z_C(i) := -(A_{II(i)}^{-1}) A_{II(i)} z_C(i)$ $O(n_i^4)$</td>
<td></td>
</tr>
</tbody>
</table>
Weak scalability for 3D Laplacian

Target machine: HELIOS@IFERC-CSC
4,410 bullx B510 compute blades (2 Intel Xeon E5-2680 8-core CPUs; 64GB)

• Target problem: \(-\Delta u = f\) on \(\overline{\Omega} = [0, 2] \times [0, 1] \times [0, 1]\)

• Uniform global mesh (Q1 FEes) + Uniform partition (cubic local meshes)

• 8, 432, \ldots, 27648 cores for fine duties

• Direct solution of Dirichlet/Neumann/coarse problems (PARDISO)

• Entire 16-core blade for coarse-grid duties (multi-threaded PARDISO)

• Gradually larger local problem sizes: \(\frac{H}{h} = 30^3, 40^3\) FEes/core
Weak scaling 2-level BDDC

3D Laplacian problem on HELIOS@IFERC
4,410 bullx B510 compute blades (2 Intel Xeon E5-2680 8-core CPUs; 64GB)

Weak scaling for BDDC(ce)

<table>
<thead>
<tr>
<th>#cores</th>
<th>H/h³=30³ (27K) FEs/core</th>
<th>H/h³=40³ (64K) FEs/core</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.5K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.7K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21.3K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27.6K</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Weak scaling for BDDC(cef)

<table>
<thead>
<tr>
<th>#cores</th>
<th>H/h³=30³ (27K) FEs/core</th>
<th>H/h³=40³ (64K) FEs/core</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.5K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.7K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21.3K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27.6K</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total time (secs.)

BDDC(ce) BDDC(cef)
Why BDDC for extreme scales?

1. (Mathematically supported) **extremely aggressive coarsening**
   
   \((10^5 - 10^6\) size reduction between fine/coarse level)

2. The coarse matrix has a similar **sparsity** as the original matrix

3. Coarse/local components can be computed **in parallel** (like additive)

4. ALL local + coarse problems can be solved **inexactly** (AMG-cycle)

5. A **multilevel** extension is possible (for extreme core counts)

- Let us see how to exploit (4), in order to boost scalability further, **reduce memory requirements, linear complexity** (energy-reducing) (**overlapped/inexact** implementation)

- Weakly scalable algorithm for AMG-cycle + additional **null space**
  
  preservation required (simple/cheap modification) [Dohrmann, 2007]
Weak scalability for 3D Laplacian

Target machine: JUQUEEN@JSC
28,672 compute nodes (16-core, 64-way threaded IBM PPC A2; 16 GB)

- Target problem: \(-\Delta u = f\) on \(\bar{\Omega} = [0, 2] \times [0, 1] \times [0, 1]\)
- Uniform global mesh (Q1 FEs) + Uniform partition (cubic local meshes)
- 8, 432, \ldots, 93312 cores for fine duties
- Serial AMG preconditioners (HSL_MI20)
- 1 core for coarse-grid duties
- Fixed local problem size \(\frac{H}{h} = 60^3\) FEs/core
Weak scaling 2-level Inexact BDDC(ce)

3D Laplacian problem on IBM BG/Q (JUQUEEN@JSC)

\[ \frac{H}{h} = 60 \] (216K FEs/core)

14% efficiency loss on 93.3K cores
Only 1 core for coarse duties (!)
Largest problem: 20.2 billion DoFs

<table>
<thead>
<tr>
<th>Outer solver</th>
<th>Φ</th>
<th>Dirichlet</th>
<th>Neumann</th>
<th>Coarse</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCG</td>
<td>AMG(2)</td>
<td>AMG(1)</td>
<td>AMG(2)</td>
<td>AMG(1)</td>
</tr>
</tbody>
</table>

Memory usage:
- Fine-grid cores: 538.6MB (< 1GB)
- Coarse-grid core (93.3K cores): 582.7MB (< 1GB)
Outline

1. Introduction and motivation
2. BDDC preconditioner
3. (Exact/Inexact) BDDC MPI-parallel implementation
4. MultiLevel BDDC MPI-parallel implementation
5. Conclusions and future work
Why BDDC for extreme scales?

1. (Mathematically supported) extremely aggressive coarsening (10^5 \(-\) 10^6 size reduction between fine/coarse level)

2. The coarse matrix has a similar sparsity as the original matrix

3. Coarse/local components can be computed in parallel (like additive)

4. ALL local + coarse problems can be solved inexactly (AMG-cycle)

5. A multilevel extension is possible (for extreme core counts)

- (1)-(2)-(3)-(4) already exploited in our codes
- Let us see how to exploit (5), in order to boost scalability even further (for exact version for the moment)
MLBDDC basic idea

MLBDDC [Mandel et. al., 2008]: Replace coarse problem by BDDC precond

\[ V_h^1 \quad \overrightarrow{I_1^t} \quad \overrightarrow{I_1} \quad \tilde{V}_h^1 \]

LEVEL 1
MLBDDC basic idea

MLBDDC [Mandel et. al., 2008]: Replace coarse problem by BDDC precond

LEVEL 1

LEVEL 2
Highly scalable implementation of MLBDDC

Naive parallel implementation

Overlapped implementation

Our approach
Goal: strike a balance such that blue/red areas are kept below green ones!

<table>
<thead>
<tr>
<th>L1 MPI tasks</th>
<th>L2 MPI tasks</th>
<th>L3 MPI task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify local coarse DoFs</td>
<td>Gather coarse-grid DoFs</td>
<td>Build coarse-grid DoFs</td>
</tr>
</tbody>
</table>

Algorithm 1 (k \(\equiv i_L\))
- \(A_{C}^{(u_2)} := \text{assemb}(A_C^{(u_1)})\)
- Compute \(\Phi_{i_2}\)
- Gather \(A_C^{(u_2)}\)

Algorithm 2 (k \(\equiv i_L\))
- \(A_{C}^{(u_2)} := \text{assemb}(A_C^{(u_1)})\)
- Identify local coarse DoFs

Algorithm 3 (k \(\equiv i_L\))
- \(A_{C}^{(u_2)} := \text{assemb}(A_C^{(u_1)})\)
- Gather \(A_C^{(u_2)}\)

Algorithm 4 (k \(\equiv i_L\))
- \(r_C^{(u_2)} := \text{assemb}(r_C^{(u_1)})\)
- Gather \(r_C^{(u_2)}\)

Algorithm 5 (k \(\equiv i_L\))
- \(r_C^{(u_2)} := \text{assemb}(r_C^{(u_1)})\)
- Gather \(r_C^{(u_2)}\)

Algorithm 6 (k \(\equiv i_L\))
- Scatter \(z_C\) into \(z_C^{(u_2)}\)

Algorithm 1
- \(\text{Re+Sy fact}(G_{AF}^{(k)})\)
- \(\text{Re+Sy fact}(G_{A_H}^{(k)})\)

Algorithm 2
- \(\text{Num fact}(A_F^{(k)})\)

Algorithm 3
- \(\text{Num fact}(A_{II}^{(k)})\)

Algorithm 4
- \(\delta_{ij}^{(k)} := (A_{ij}^{(k)})^{-1}r_{ij}^{(k)}\)
- \(t_{ij}^{(k)} := r_{ij}^{(k)} - A_{ij}^{(k)}\delta_{ij}^{(k)}\)
- \(r^{(k)} := l_{ij}^{(k)}\)

Algorithm 5
- Solve \(A_F^{(k)} \left[ \begin{array}{c} t \\ s_F^{(k)} \end{array} \right] = \left[ \begin{array}{c} r^{(k)} \\ 0 \end{array} \right]\)

Algorithm 6
- \(s_C^{(k)} := \Phi_{ij} z_C^{(k)}\)
- \(z^{(k)} := t_{ij} s_F^{(k)} + s_C^{(k)}\)
- \(z_t^{(k)} := -A_{II}^{(k)} z_F^{(k)}\)
- \(z_{ij}^{(k)} := z_t^{(k)} + \delta_{ij}^{(k)}\)
Weak scaling 3-lev BDDC(ce) solver

3D Laplacian problem on IBM BG/Q (JUQUEEN@JSC)
16 MPI tasks/compute node, 1 OpenMP thread/MPI task

Experiment set-up

<table>
<thead>
<tr>
<th>Lev.</th>
<th># MPI tasks</th>
<th>FEs/core</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>42.8K 74.1K 117.6K 175.6K 250K 343K 456.5K</td>
<td>20³/25³/30³/40³</td>
</tr>
<tr>
<td>2nd</td>
<td>125 216 343 512 729 1000 1331</td>
<td>7³</td>
</tr>
<tr>
<td>3rd</td>
<td>1 1 1 1 1 1 1</td>
<td>n/a</td>
</tr>
</tbody>
</table>
Weak scaling 3-lev BDDC(cef) solver

3D Laplacian problem on IBM BG/Q (JUQUEEN@JSC)
16 MPI tasks/compute node, 1 OpenMP thread/MPI task

Experiment set-up

<table>
<thead>
<tr>
<th>Lev.</th>
<th># PCG iterations</th>
<th>Total time (secs.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3rd</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Weak scaling for 4-level BDDC(ce) solver with $H_2/h_2=4$, $H_3/h_3=3$

3D Laplacian problem on IBM BG/Q (JUQUEEN@JSC)
64 MPI tasks/compute node, 1 OpenMP thread/MPI task

<table>
<thead>
<tr>
<th>Lev.</th>
<th># MPI tasks</th>
<th>FEs/core</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>110.6K</td>
<td>$10^3$</td>
</tr>
<tr>
<td></td>
<td>216K</td>
<td>$20^3$</td>
</tr>
<tr>
<td></td>
<td>373.2K</td>
<td>$25^3$</td>
</tr>
<tr>
<td>2nd</td>
<td>1.73K</td>
<td>$4^3$</td>
</tr>
<tr>
<td></td>
<td>3.38K</td>
<td>$3^3$</td>
</tr>
<tr>
<td></td>
<td>5.83K</td>
<td>n/a</td>
</tr>
<tr>
<td>3rd</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>125</td>
<td></td>
</tr>
<tr>
<td></td>
<td>216</td>
<td></td>
</tr>
<tr>
<td></td>
<td>343</td>
<td></td>
</tr>
<tr>
<td></td>
<td>512</td>
<td></td>
</tr>
<tr>
<td></td>
<td>729</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1K</td>
<td></td>
</tr>
<tr>
<td>4th</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

# PCG iters.

Total time (secs.)
Weak scaling 4-lev BDDC(cef) + 4 MPI tasks/core

3D Laplacian problem on IBM BG/Q (JUQUEEN@JSC)
64 MPI tasks/compute node, 1 OpenMP thread/MPI task
Weak scaling 3-lev BDDC(ce) solver

3D Linear Elasticity problem on IBM BG/Q (JUQUEEN@JSC)
16 MPI tasks/compute node, 1 OpenMP thread/MPI task

#PCG iterations  Total time (secs.)

<table>
<thead>
<tr>
<th>Lev.</th>
<th># MPI tasks</th>
<th>FEs/core</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>42.8K 74.1K 117.6K 175.6K 250K 343K 456.5K</td>
<td>15^3/20^3/25^3</td>
</tr>
<tr>
<td>2nd</td>
<td>125 216 343 512 729 1000 1331</td>
<td>7^3</td>
</tr>
<tr>
<td>3rd</td>
<td>1 1 1 1 1 1 1</td>
<td>n/a</td>
</tr>
</tbody>
</table>
3D Linear Elasticity problem on IBM BG/Q (JUQUEEN@JSC)
16 MPI tasks/compute node, 1 OpenMP thread/MPI task
Outline

1. Introduction and motivation
2. BDDC preconditioner
3. (Exact/Inexact) BDDC MPI-parallel implementation
4. MultiLevel BDDC MPI-parallel implementation
5. Conclusions and future work
Conclusions and future work

- Highly scalable implementation of (exact) MLBDDC
  - Fully-distributed
  - Communicator-aware
  - Interlevel-overlapped (coarse-grain comput./comput./comm. overlap)
  - Recursive (extensible to arbitrary # levels)

- Remarkable scalability
  - 3D Laplacian and Linear Elasticity PDEs
  - 3/4 levels are sufficient to (efficiently) scale till full JUQUEEN
  - Largest scaling/problem sizes reported so far with (exact) DDM

- Future work includes:
  - Richardson/Krylov subspace corrections on intermediate levels
  - Inexact MLBDDC (hybrid DD-AMG)
  - (Non-naive) Hybrid MPI+threads implementation
Conclusions and future work

- Highly scalable implementation of (exact) MLBDDC
  - Fully-distributed
  - Communicator-aware
  - Interlevel-overlapped (coarse-grain comput./comput./comm. overlap)
  - Recursive (extensible to arbitrary # levels)

- Remarkable scalability
  - 3D Laplacian and Linear Elasticity PDEs
  - 3/4 levels are sufficient to (efficiently) scale till full JUQUEEN
  - Largest scaling/problem sizes reported so far with (exact) DDM

- Future work includes:
  - Richardson/Krylov subspace corrections on intermediate levels
  - Inexact MLBDDC (hybrid DD-AMG)
  - (Non-naive) Hybrid MPI+threads implementation
Conclusions and future work

- Highly scalable implementation of (exact) MLBDDC
  - Fully-distributed
  - Communicator-aware
  - Interlevel-overlapped (coarse-grain comput./comput./comm. overlap)
  - Recursive (extensible to arbitrary # levels)

- Remarkable scalability
  - 3D Laplacian and Linear Elasticity PDEs
  - 3/4 levels are sufficient to (efficiently) scale till full JUQUEEN
  - Largest scaling/problem sizes reported so far with (exact) DDM

- Future work includes:
  - Richardson/Krylov subspace corrections on intermediate levels
  - Inexact MLBDDC (hybrid DD-AMG)
  - (Non-naive) Hybrid MPI+ threads implementation
FEMPAR: Massively parallel finite element simulation of multiphysics problems

- Development centered around (although not limited to) computational fusion → highly coupled, highly non-linear, multiphysics problems

Based on 3 main pillars:

- Advanced stabilized FEM formulations (equal-order interpolation for all unknowns)
- Scalable “physics-based” block preconditioning of fully-coupled, implicit linear system
- Balancing (coarse correction) domain decomposition for one-physics diagonal blocks

Software characteristics:

- Open Source – GNU GPLv3 (available soon)
- Fortran 90/95/2003 (OO programming style)
- MPI, OpenMP (actually provided by external solver libraries)
- Highly-scalable implementation of MLBDDC
- Tested on x86 (HELIOS), Cray XE6 (HERMIT), IBM BG/Q platforms
- Scalability: 458,752 cores on JUQUEEN (High-Q club status, 2014)
Thank you!


Preprints available at http://badia.rmee.upc.edu/sbadia_ar.html

LSSC team: https://web.cimne.upc.edu/groups/comfus/

Work funded by the European Research Council under Starting Grant 258443 - COMFUS: Computational Methods for Fusion Technology