

Fast Graph Partitioning and Applications to Preconditioning

David Fritzsche^{*,†}, Andreas Frommer^{*}, Daniel B. Szyld[†]

* Bergische Universität Wuppertal

† Temple University

SIAM Workshop on Combinatorial Scientific Computing (CSC09)

October 29-31, 2009

Monterey, USA



Outline

Goal: Solve linear system

$$Ax = b$$

where $A \in \mathbb{R}^{n \times n}$, nonsingular, sparse and non-symmetric.

- ▶ non-overlapping partitions & applications to preconditioning
- ▶ Schwarz methods — why adding overlap is a jolly idea
- ▶ A fast algorithm for finding overlapping subgraphs
- ▶ Numerical Experiments



Non-overlapping Partition

- ▶ Let $G = (V, E)$ be the directed graph of A
- ▶ $\{V_i\}_{i=1, \dots, q}$ (non-overlapping) partition of V , $|V_i| = n_i$
- ▶ $R_i \in \mathbb{R}^{n_i \times n}$ restriction ($R_i = I_n(V_i, :)$ in Matlab notation)

Partition $\{V_i\}$ non-overlapping \Rightarrow we can reorder A s.t.

$$PAP^T = \begin{bmatrix} A_{11} & \cdots & A_{1q} \\ \vdots & & \vdots \\ A_{q1} & \cdots & A_{qq} \end{bmatrix}$$

where A_{ij} corresponds to V_i in the sense that $G(A_{ij}) = G|_{V_i}$

Instead: $A_{ij} := R_i A R_j^T$, $A_{ij} \in \mathbb{R}^{n_i \times n_j}$
 $A_i := A_{ii}$, $x_i := R_i x$, $b_i := R_i b$

Block Jacobi / Block Gauss-Seidel

Classical block Jacobi method:

```
for  $k = 1, 2, \dots$  do  
  for  $i = 1, \dots, q$  do  
     $x_i^{(k+1)} := A_i^{-1} \left( b_i - \sum_{j \neq i} A_{ij} x_j^{(k)} \right)$   
  end for  
end for
```

Classical block Gauss-Seidel method:

```
for  $k = 1, 2, \dots$  do  
  for  $i = 1, \dots, q$  do  
     $x_i^{(k+1)} := A_i^{-1} \left( b_i - \sum_{j < i} A_{ij} x_j^{(k+1)} - \sum_{j > i} A_{ij} x_j^{(k)} \right)$   
  end for  
end for
```

Block Jacobi

$$D := \sum R_i^T A_i R_i \left(= \begin{bmatrix} A_1 & & 0 \\ & \ddots & \\ 0 & & A_q \end{bmatrix} \text{ if } A \text{ is reordered} \right)$$

$$D^{-1} := \sum R_i^T A_i^{-1} R_i \left(= \begin{bmatrix} A_1^{-1} & & 0 \\ & \ddots & \\ 0 & & A_q^{-1} \end{bmatrix} \text{ if } A \text{ is reordered} \right)$$

Block Jacobi written as stationary iteration:

$$x^{(k+1)} := D^{-1}(D - A)x^{(k)} + D^{-1}b$$

Preconditioning: $M^{-1} := D^{-1}$



Block Gauss-Seidel

Notation:

- ▶ $\Pi_i := R_i^T A_i^{-1} R_i A$
- ▶ $Q_q := (I - \Pi_q) \cdots (I - \Pi_1)$

Block Gauss-Seidel written as stationary iteration:

$$x^{(k+1)} := Q_q x^{(k)} + (I - Q_q) A^{-1} b$$

Preconditioning: $M^{-1} := (I - Q_q) A^{-1}$



A different view...

For now: Just block Gauss-Seidel

Before:

for $i=1, \dots, q$ **do**

$$x_i^{(k+1)} := A_i^{-1} \left(b_i - \sum_{j < i} A_{ij} x_j^{(k+1)} - \sum_{j > i} A_{ij} x_j^{(k)} \right)$$

end for

Rewritten:

$$z := x^{(k)}$$

for $i=1, \dots, q$ **do**

$$z_i := A_i^{-1} \left(b_i - \sum_{j < i} A_{ij} z_j - \sum_{j > i} A_{ij} z_j \right)$$

end for

$$x^{(k+1)} := z$$



A different view...

For now: Just block Gauss-Seidel

Before:

for $i=1, \dots, q$ **do**

$$x_i^{(k+1)} := A_i^{-1} \left(b_i - \sum_{j < i} A_{ij} x_j^{(k+1)} - \sum_{j > i} A_{ij} x_j^{(k)} \right)$$

end for

Rewritten:

$$z := x^{(k)}$$

for $i=1, \dots, q$ **do**

$$z_i := A_i^{-1} \left(b_i + A_i z_i - \sum_{j=1}^q A_{ij} z_j \right)$$

end for

$$x^{(k+1)} := z$$



A different view...

For now: Just block Gauss-Seidel

Before:

for $i=1, \dots, q$ **do**

$$x_i^{(k+1)} := A_i^{-1} \left(b_i - \sum_{j < i} A_{ij} x_j^{(k+1)} - \sum_{j > i} A_{ij} x_j^{(k)} \right)$$

end for

Rewritten:

$$z := x^{(k)}$$

for $i=1, \dots, q$ **do**

$$z_i := z_i + A_i^{-1} R_i (b - Az)$$

end for

$$x^{(k+1)} := z$$



A different view...

For now: Just block Gauss-Seidel

Before:

for $i=1, \dots, q$ **do**

$$x_i^{(k+1)} := A_i^{-1} \left(b_i - \sum_{j < i} A_{ij} x_j^{(k+1)} - \sum_{j > i} A_{ij} x_j^{(k)} \right)$$

end for

Rewritten:

$$z := x^{(k)}$$

for $i=1, \dots, q$ **do**

$$z := z + \underbrace{R_i^T A_i^{-1} R_i (b - Az)}_{\text{local error}}$$

end for

$$x^{(k+1)} := z$$



Block Gauss-Seidel iteration:

$$z := x^{(k)}$$

for $i=1, \dots, q$ **do**

$$z := z + R_i^T \underbrace{A_i^{-1} R_i (b - Az)}_{\text{local error}}$$

end for

$$x^{(k+1)} := z$$

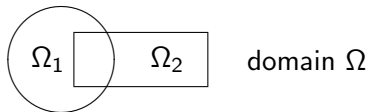
Is it necessary for V_i to be non-overlapping? **NO**

It may actually help if the V_i **do** overlap...

Schwarz Alternating Method

Schwarz introduced in 1870 an *alternating method* for solving partial differential equations on irregular domains.

Setting: Let Ω be the union of the “regular” shaped subdomains Ω_i , $i = 1, \dots, m$



Schwarz alternating method:

- ▶ Solve the PDE on Ω_i , $i = 1, \dots, m$
For $\partial\Omega_i \setminus \partial\Omega$ use interior values of a different subdomain
- ▶ Repeat until convergence

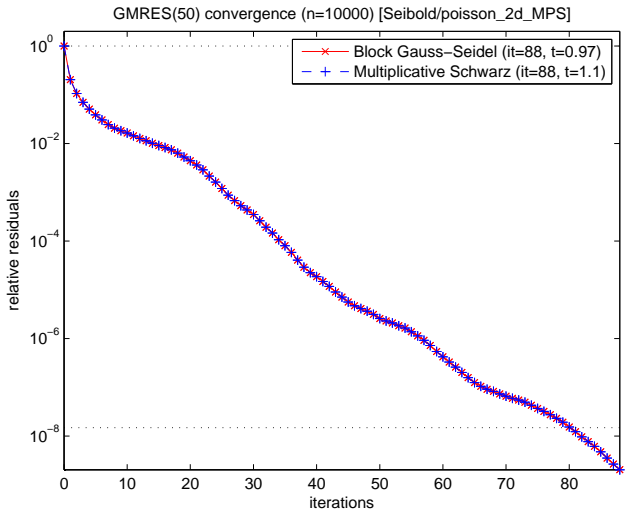


Multiplicative / Additive Schwarz Method

overlapping Jacobi = additive Schwarz

overlapping Gauss-Seidel = multiplicative Schwarz

Multiplicative Schwarz w/o Overlap = Block Gauss-Seidel





Multiplicative Schwarz Preconditioning

Left preconditioning: $A \rightarrow M^{-1}A$

Here: $M^{-1} = (I - Q_q)A^{-1}$

$\Rightarrow A \rightarrow I - Q_q$ (useful for writing an optimized preconditioned matrix-vector multiplication, i.e., $z := A^{-1}Av$)

For Matlab we need just application of preconditioner M^{-1} :

input: vector v

output: $z := M^{-1}v$ $\{z := (I - Q_m)A^{-1}v\}$

$z := 0$

for $i = 1, \dots, q$ **do**

$z := z + (R_i^T A_i^{-1} R_i)(v - Az)$

end for

OBGP

OBGP = **O**verlapping **B**locks by **G**rowing a **P**artition

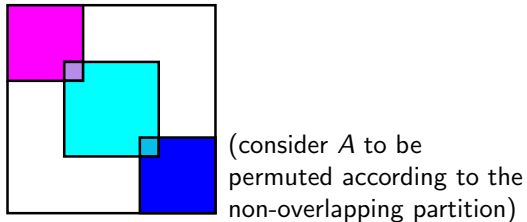
Overview:

- ▶ Find a partition (Metis, PABLO, ...)
- ▶ Take the blocks and grow them to add some overlap

What nodes to add?

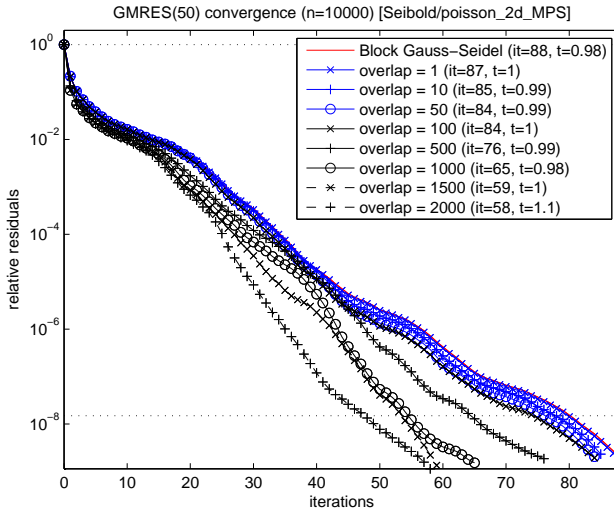
Problem: Not all Overlap is good Overlap

Idea: Add to each diagonal block A_i some rows/columns of the neighboring blocks.



↪ no or only small improvement in many cases

Adding rows/columns

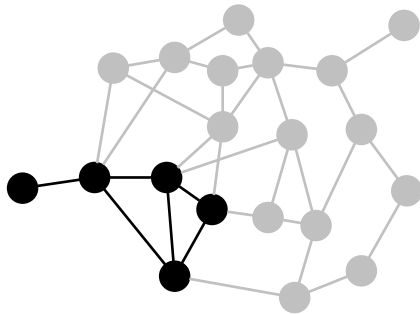


Level Sets

Solution: Add **level sets!**

Definition: Let $G = (V, E)$ be a graph, $V_i \subset V$. The first *level set* around V_i is the set of all nodes adjacent to V_i , but not in V_i , i.e.,

$$L(V_i) := \{v \in V \setminus V_i \mid v \text{ adjacent to } V_i\}.$$

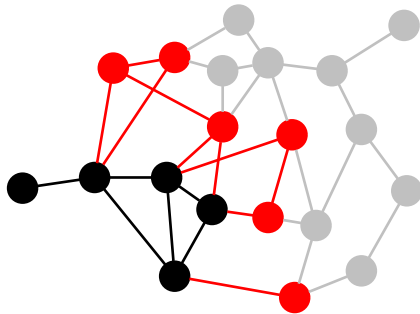


Level Sets

Solution: Add **level sets!**

Definition: Let $G = (V, E)$ be a graph, $V_i \subset V$. The first *level set* around V_i is the set of all nodes adjacent to V_i , but not in V_i , i.e.,

$$L(V_i) := \{v \in V \setminus V_i \mid v \text{ adjacent to } V_i\}.$$

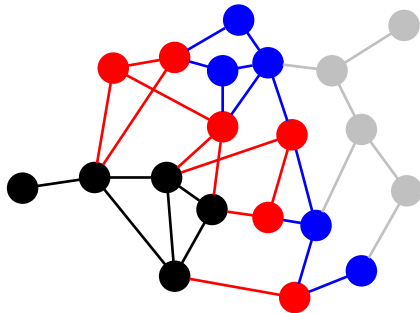


Level Sets

Solution: Add **level sets!**

Definition: Let $G = (V, E)$ be a graph, $V_i \subset V$. The first *level set* around V_i is the set of all nodes adjacent to V_i , but not in V_i , i.e.,

$$L(V_i) := \{v \in V \setminus V_i \mid v \text{ adjacent to } V_i\}.$$

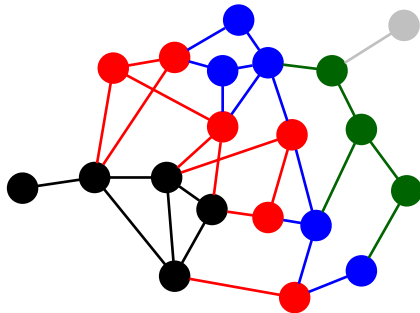


Level Sets

Solution: Add **level sets!**

Definition: Let $G = (V, E)$ be a graph, $V_i \subset V$. The first *level set* around V_i is the set of all nodes adjacent to V_i , but not in V_i , i.e.,

$$L(V_i) := \{v \in V \setminus V_i \mid v \text{ adjacent to } V_i\}.$$

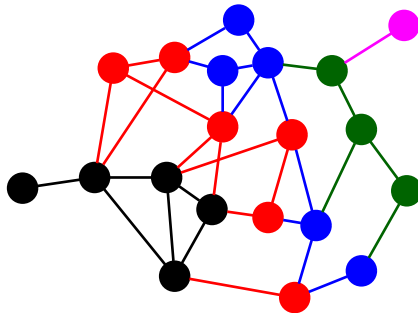


Level Sets

Solution: Add **level sets!**

Definition: Let $G = (V, E)$ be a graph, $V_i \subset V$. The first *level set* around V_i is the set of all nodes adjacent to V_i , but not in V_i , i.e.,

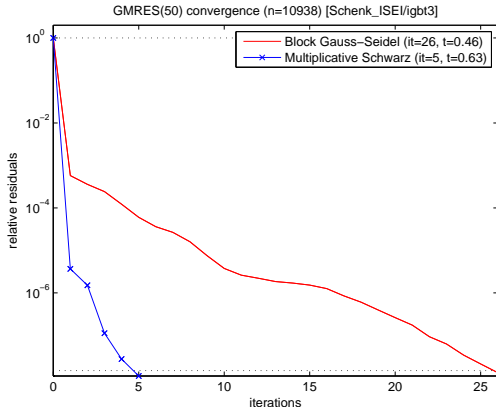
$$L(V_i) := \{v \in V \setminus V_i \mid v \text{ adjacent to } V_i\}.$$



Problem: Too much overlap

Problem: In many cases the complete level set is too large.
 \rightsquigarrow The total time for solving the system can easily grow.

Example:



Weighting Nodes

Solution

Add *partial level sets*!

Introduce node-weights and choose only nodes with large weights.

Definition: Let $B \subset V$ be a set of nodes and $i \in V$.
The weight $w(i, B)$ is defined as

$$w(i, B) := \begin{cases} \sum_{j \in B} |a_{ij}| + |a_{ji}| & \text{if } i \notin B, \\ 0 & \text{if } i \in B. \end{cases}$$

Idea: Select up to $\mu(B)^1$ nodes, choose nodes with largest weights.

¹here: $\mu(B) = \sqrt{|B|}$, motivated by being proportional to the border size of a 2D domain. Also works well in our experiments.



Summary of OBG

How OBG works

- ▶ take an existing non-overlapping partition
- ▶ grow each block B by adding nodes from $L(B)$
- ▶ add up to $\mu(B)$ nodes of largest weight to B

If more overlap is needed: Repeat \rightsquigarrow OBG(ℓ)

- ▶ Limit total growth by $\nu(B)$
 \rightsquigarrow OBG(∞) – adding partial level sets until ν is reached

Note: In many cases it is better to increase ℓ than to increase μ .



Analysis of time complexity

Recall: $\{V_i\}_{i=1,\dots,q}$ is the non-overlapping partition of V .

Without further assumptions we can prove the time complexity of $\text{OBGP}(\ell)$ to be

$$\mathcal{O}(q(\text{nnz}(\mathbf{A}) + n \log n)) \quad (\text{always})$$

If $\nu(B) \in \mathcal{O}(|B|)$ we can show the time complexity to be

$$\mathcal{O}(nd + n \log s) \quad (\nu \text{ limited})$$

where $d :=$ maximum degree in graph and $s :=$ maximum block size (before growing)

Note: The given time complexity is independent of ℓ . This is possible by using a “heap” data structure for storing the nodes in the level set. \rightsquigarrow fast selection of nodes with largest weights and fast updating between rounds.



Numerical Results

Solver: GMRES(50)

Non-Overlapping Partition: Found by PABLO

OBGP parameters:

- ▶ $\mu(B) = \sqrt{|B|}$
- ▶ $\nu(B) = \ell\sqrt{|B|}$

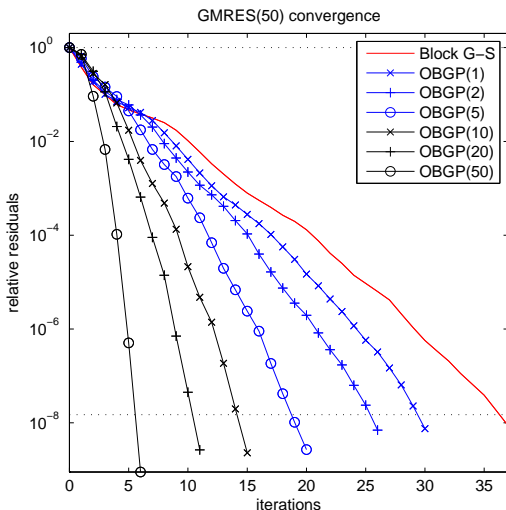
Notation:

- ▶ $V_i, i = 1, \dots, m$, are the original blocks (no overlap)
- ▶ $W_i, i = 1, \dots, m$, are the blocks after growing them (overlap)

Abbrevs. for Timing-Tables:

- ▶ P = Preprocessing + Partitioning
- ▶ LU = LU-decomposition of the preconditioner
- ▶ S = solving
- ▶ T = total time

poisson_2d_LSQ ($n = 10000$, $nnz = 187949$)

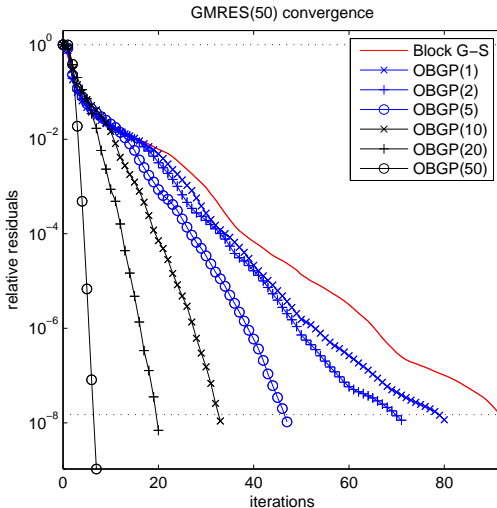


Number of blocks: 13

Method	iter	P	LU	S	T
Block G-S	37	0.12	0.11	0.49	0.72
OBGP(1)	30	0.12	0.12	0.45	0.69
OBGP(2)	26	0.12	0.13	0.38	0.63
OBGP(5)	20	0.12	0.15	0.32	0.59
OBGP(10)	15	0.12	0.2	0.26	0.58
OBGP(20)	11	0.12	0.26	0.25	0.63
OBGP(50)	6	0.12	0.61	0.29	1

Method	$\sum W_i $	$\sum nnz(W_i)$
Block G-S	10000	170914
OBGP(1)	10350	179161
OBGP(2)	10711	186436
OBGP(5)	11824	208002
OBGP(10)	13815	246596
OBGP(20)	18280	333618
OBGP(50)	35575	676142

poisson_2d_MPS ($n = 10000$, $nnz = 56915$)

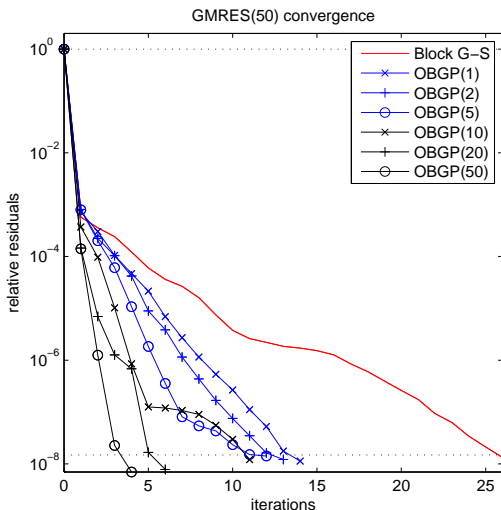


Number of blocks: 11

Method	iter	P	LU	S	T
Block G-S	92	0.044	0.048	0.92	1
OBGP(1)	80	0.044	0.057	0.86	0.96
OBGP(2)	71	0.044	0.06	0.76	0.87
OBGP(5)	47	0.044	0.073	0.56	0.68
OBGP(10)	33	0.044	0.08	0.36	0.48
OBGP(20)	20	0.044	0.11	0.24	0.39
OBGP(50)	7	0.044	0.23	0.14	0.42

Method	$\sum W_i $	$\sum nnz(W_i)$
Block G-S	10000	46242
OBGP(1)	10327	49525
OBGP(2)	10664	52170
OBGP(5)	11699	58988
OBGP(10)	13540	70505
OBGP(20)	17630	95554
OBGP(50)	33200	188884

igbt3 ($n = 10938$, $nnz = 130500$)

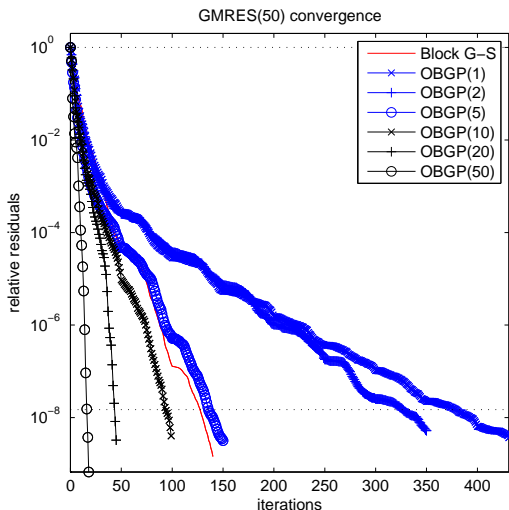


Number of blocks: 13

Method	iter	P	LU	S	T
Block G-S	26	0.082	0.093	0.29	0.46
OBGP(1)	14	0.082	0.11	0.16	0.35
OBGP(2)	13	0.082	0.11	0.16	0.35
OBGP(5)	12	0.082	0.12	0.15	0.36
OBGP(10)	11	0.082	0.14	0.17	0.39
OBGP(20)	6	0.082	0.22	0.12	0.42
OBGP(50)	4	0.082	0.47	0.16	0.71

Method	$\sum W_i $	$\sum nnz(W_i)$
Block G-S	10938	105889
OBGP(1)	11310	110580
OBGP(2)	11691	115145
OBGP(5)	12868	129025
OBGP(10)	14963	153600
OBGP(20)	19638	207899
OBGP(50)	37440	413629

garon2 ($n = 13535$, $nnz = 373235$)



Number of blocks: 15

Method	iter	P	LU	S	T
Block G-S	140	0.21	0.19	2.7	3.1
OBGP(1)	431	0.21	0.19	9.6	10
OBGP(2)	350	0.21	0.19	8	8.4
OBGP(5)	150	0.21	0.21	3.6	4
OBGP(10)	99	0.21	0.26	2.6	3.1
OBGP(20)	45	0.21	0.37	1.4	2
OBGP(50)	18	0.21	0.87	0.95	2

Method	$\sum W_i $	$\sum nnz(W_i)$
Block G-S	13535	336925
OBGP(1)	13985	350084
OBGP(2)	14448	362880
OBGP(5)	15871	399007
OBGP(10)	18400	468381
OBGP(20)	24015	622585
OBGP(50)	45360	1209050

Conclusions

- ▶ You can improve on block Gauss-Seidel preconditioning by adding overlap
- ▶ Selecting the “right” nodes for overlap is important

<http://www-ai.math.uni-wuppertal.de/~dfritzsche/>
<http://www.math.temple.edu/~daffi/>

Thank you!