

Communication-Avoiding Incomplete LU0 (CA-ILU0) Preconditioner

Sophie Moufawad[†], Laura Grigori[†]

[†]: Institut National de Recherche en Informatique et Automatique (INRIA),
Rocquencourt

Sparse Days Meeting 2013 at CERFACS, Toulouse
June 17-18, 2013

Motivation

- ▶ Cost of Algorithm: Arithmetic operations and Communication

Motivation

- ▶ Cost of Algorithm: Arithmetic operations and Communication
- ▶ Bottleneck communication \implies Communication-Avoiding methods

Motivation

- ▶ Cost of Algorithm: Arithmetic operations and Communication
- ▶ Bottleneck communication \implies Communication-Avoiding methods
- ▶ Interested in solving $Ax = b$

Motivation

- ▶ Cost of Algorithm: Arithmetic operations and Communication
- ▶ Bottleneck communication \implies Communication-Avoiding methods
- ▶ Interested in solving $Ax = b$
- ▶ CA-Iterative Methods based on **s-step methods**:
CA-GMRES, CA-CG

Motivation

- ▶ Cost of Algorithm: Arithmetic operations and Communication
- ▶ Bottleneck communication \implies Communication-Avoiding methods
- ▶ Interested in solving $Ax = b$
- ▶ CA-Iterative Methods based on **s-step methods**:
CA-GMRES, CA-CG
- ▶ Few CA-Preconditioners !!

Goal

- Design Communication Avoiding preconditioners M that
 1. Speed up the convergence of the system $Ax = b$

Goal

- Design Communication Avoiding preconditioners M that
 1. Speed up the convergence of the system $Ax = b$
 2. enhance its communication avoiding parallelizability

Goal

- Design Communication Avoiding preconditioners M that
 1. Speed up the convergence of the system $Ax = b$
 2. enhance its communication avoiding parallelizability
- The “Communication Avoiding” preconditioned system $M_{ca}^{-1}Ax = M_{ca}^{-1}b$ should

Goal

- Design Communication Avoiding preconditioners M that
 1. Speed up the convergence of the system $Ax = b$
 2. enhance its communication avoiding parallelizability
- The “Communication Avoiding” preconditioned system $M_{ca}^{-1}Ax = M_{ca}^{-1}b$ should
 1. have the same order of convergence as the preconditioned system $M^{-1}Ax = M^{-1}b$

Goal

- Design Communication Avoiding preconditioners M that
 1. Speed up the convergence of the system $Ax = b$
 2. enhance its communication avoiding parallelizability
- The “Communication Avoiding” preconditioned system $M_{ca}^{-1}Ax = M_{ca}^{-1}b$ should
 1. have the same order of convergence as the preconditioned system $M^{-1}Ax = M^{-1}b$
 2. need less communication

Goal

- Design Communication Avoiding preconditioners M that
 1. Speed up the convergence of the system $Ax = b$
 2. enhance its communication avoiding parallelizability
- The “Communication Avoiding” preconditioned system $M_{ca}^{-1}Ax = M_{ca}^{-1}b$ should
 1. have the same order of convergence as the preconditioned system $M^{-1}Ax = M^{-1}b$
 2. need less communication
- ▶ Present Communication Avoiding ILU(0) preconditioner (CA-ILU0)

Introduction

Preconditioned Krylov Subspace Methods

$s - step$ GMRES

CA-GMRES

The Matrix Powers Kernel

ILU Matrix Powers Kernel

ILU Preconditioner

ILU Matrix Powers Kernel

Reordering the matrix A based on Nested Dissection

Communication Avoiding ILU0 (CA-ILU0(s)) reordering

The Complexity of the CA-ILU0(s) reordering

The Expected Performance of ND CA-ILU0 Preconditioner

Reordering the matrix A based on k-way Partitioning

Communication Avoiding ILU0 (CA-ILU0(s)) reordering

The Expected Performance of k-way CA-ILU0 Preconditioner

Conclusion, Current and Future Work

GMRES

- ▶ is a Krylov subspace method developed by Saad and Schultz in 1986
- ▶ finds the solution of the system $Ax = b$ by minimizing the residual at each iteration

$$\|r_k\| = \|b - Ax_k\| = \min\|b - Ax\|, \forall x \in x_0 + \kappa_k(A, r_0)$$

GMRES

- ▶ is a Krylov subspace method developed by Saad and Schultz in 1986
- ▶ finds the solution of the system $Ax = b$ by minimizing the residual at each iteration

$$\|r_k\| = \|b - Ax_k\| = \min\|b - Ax\|, \forall x \in x_0 + \kappa_k(A, r_0)$$

- ▶ builds up an orthonormal basis q_1, q_2, \dots, q_k for $\kappa_k(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \dots, A^{k-1}r_0\}$ using the Arnoldi procedure which is a Modified Gram-Schmidt procedure.
- ▶ solves a least square problem using Givens rotations and Backward substitution.

GMRES ($x_0, A, b, \epsilon, k_{max}$)

Compute $r_0 = b - Ax_0$, $\rho = ||r_0||_2$, $\beta = \rho$, $q_1 = \frac{r_0}{\rho}$, $P_0 = \beta e_1$, $k = 1$

While ($\rho \geq \epsilon ||b||_2$ and $k < k_{max}$) Do

2. Perform an Arnoldi iteration, generate the $(k + 1)^{th}$ vector of the Arnoldi basis Q_{k+1} and the k^{th} column of the Upper Hessenberg matrix H_k

Compute $q_{j+1} = Aq_j$

for $j = 1, \dots, k$

$h_{j,k} = q_j^T q_{k+1}$ and $q_{k+1} = q_{k+1} - h_{j,k} q_j$

end

If $q_{k+1} = 0$ Stop

Else $h_{k+1,k} = ||q_{k+1}||_2$

$$q_{k+1} = \frac{q_{k+1}}{h_{k+1,k}}$$

3. Minimize $||\beta e_1 - H_k y||$ over \mathbb{R}^k to obtain y_k using:

a. Givens rotations that transform H_k into an upper triangular system

b. Backward substitution on the system $H_k y = P_k$ where P_k is the transformation of βe_1 under the Givens rotation

4. $\rho = ||P_k - H_k y_k||_2 = |P_k(k+1)|$, $k = k + 1$

end

$$x = x_0 + Q_{k-1} y_{k-1}$$

Preconditioned GMRES ($x_0, M, A, b, \epsilon, k_{max}$)

Compute $r_0 = M^{-1}(b - Ax_0)$, $\rho = \|r_0\|_2$, $\beta = \rho$, $q_1 = \frac{r_0}{\rho}$, $P_0 = \beta e_1$, $k = 1$

While ($\rho \geq \epsilon \|b\|_2$ and $k < k_{max}$) Do

 2. Perform an Arnoldi iteration, generate the $(k + 1)^{th}$ vector of the Arnoldi

 basis Q_{k+1} and the k^{th} column of the Upper Hessenberg matrix H_k

 Compute $q_{j+1} = M^{-1}Aq_j$

 for $j = 1, \dots, k$

$h_{j,k} = q_j^T q_{k+1}$ and $q_{k+1} = q_{k+1} - h_{j,k} q_j$

 end

 If $q_{k+1} = 0$ Stop

 Else $h_{k+1,k} = \|q_{k+1}\|_2$

$$q_{k+1} = \frac{q_{k+1}}{h_{k+1,k}}$$

 3. Minimize $\|\beta e_1 - H_k y\|$ over \mathbb{R}^k to obtain y_k using:

 a. Givens rotations that transform H_k into an upper triangular system

 b. Backward substitution on the system $H_k y = P_k$ where P_k is the

 transformation of βe_1 under the Givens rotation

 4. $\rho = \|P_k - H_k y_k\|_2 = |P_k(k+1)|$, $k = k + 1$

end

$$x = x_0 + Q_{k-1}y_{k-1}$$

$s - step$ GMRES

s-step GMRES ($x_0, \mathbf{A}, \mathbf{b}, \epsilon, s$)

1. Compute $r_0 = b - Ax_0$, $q_1 = r_0 / \|r_0\|_2$
2. Perform an **Arnoldi(s)** iteration
 - i. Compute $q_2 = Ar_0, q_3 = A^2r_0, q_4 = A^3r_0, \dots, q_{s+1} = A^s r_0$
where $q_i = Aq_{i-1}$
 - ii. Orthogonalize q_1, q_2, \dots, q_{s+1} using a QR factorization
 - iii. Reconstruct the upper Hessenberg matrix H_s
3. Solve the Least Square problem $y_s = \min_y \|P_s - H_s y\|_2$
4. $\rho = \|P_s - H_s y_s\|_2$
5. $x_s = x_0 + Q_s y_s$
6. if ($\rho \geq \epsilon \|b\|_2$)
 - Let $x_0 = x_s$ and call s-step GMRES ($x_s, \mathbf{A}, \mathbf{b}, \epsilon, s$)
 - else
 - x_s is the approximate solution

CA-GMRES ($x_0, \mathbf{A}, \mathbf{b}, \epsilon, s, l$)

1. Compute $r_0 = b - Ax_0$, $q_1 = r_0 / \|r_0\|_2$, $\rho = \|r_0\|_2$, $i = 0$
2. Perform an Arnoldi(s, l) iteration

While ($\rho \geq \epsilon \|b\|_2$ and $i < l$)

- i. Compute $q_{si+2}, q_{si+3}, q_{si+4}, \dots, q_{si+(s+1)}$ using Matrix Powers Kernel
- ii. Orthogonalize q_{si+j} ($2 \leq j \leq s+1$) against q_j ($2 \leq j \leq si+1$) using Block Gram Schmidt
- iii. Orthogonalize $q_{si+1}, q_{si+2}, \dots, q_{si+(s+1)}$ using a TSQR factorization
- iv. Reconstruct the upper Hessenberg matrix H_s
- v. Update ρ , $i = i + 1$

3. Solve the Least Square problem $y_{si} = \min_y \|P_s - H_s y\|_2$

5. $x_{si} = x_0 + Q_s y_{si}$

6. if ($\rho \geq \epsilon \|b\|_2$)

Let $x_0 = x_{si}$ and call CA-GMRES ($x_s, \mathbf{A}, \mathbf{b}, \epsilon, s, l$)

else

x_{si} is the approximate solution

The Parallel $s - step$ Matrix-Vector Multiplication (Matrix Powers Kernel)

The $s - step$ matrix-Vector multiplication consists of performing s matrix vector multiplications at a time Ax till $A^s x$ in parallel by p processors where

- ▶ the graph of A , $G(A)$, is partitioned into p subgraphs using known partitioning techniques

The Parallel $s - step$ Matrix-Vector Multiplication (Matrix Powers Kernel)

The $s - step$ matrix-Vector multiplication consists of performing s matrix vector multiplications at a time Ax till $A^s x$ in parallel by p processors where

- ▶ the graph of A , $G(A)$, is partitioned into p subgraphs using known partitioning techniques
- ▶ each processor fetches all the data needed from the beginning

The Parallel $s - step$ Matrix-Vector Multiplication (Matrix Powers Kernel)

The $s - step$ matrix-Vector multiplication consists of performing s matrix vector multiplications at a time Ax till $A^s x$ in parallel by p processors where

- ▶ the graph of A , $G(A)$, is partitioned into p subgraphs using known partitioning techniques
- ▶ each processor fetches all the data needed from the beginning
- ▶ each processor computes a part of Ax till $A^s x$ without any communication with other processors by performing extra redundant flops

1	2	3	4	5	6	7	8	9	10	51	52	53	54	55	56	57	58	59	60
11	12	13	14	15	16	17	18	19	20	61	62	63	64	65	66	67	68	69	70
21	22	23	24	25	26	27	28	29	30	71	72	73	74	75	76	77	78	79	80
31	32	33	34	35	36	37	38	39	40	81	82	83	84	85	86	87	88	89	90
41	42	43	44	45	46	47	48	49	50	91	92	93	94	95	96	97	98	99	100

101 102 103 104 105 106 107 108 109 110 151 152 153 154 155 156 157 158 159 160

111 112 113 114 115 116 117 118 119 120 161 162 163 164 165 166 167 168 169 170

121 122 123 124 125 126 127 128 129 130 171 172 173 174 175 176 177 178 179 180

131 132 133 134 135 136 137 138 139 140 181 182 183 184 185 186 187 188 189 190

141 142 143 144 145 146 147 148 149 150 191 192 193 194 195 196 197 198 199 200



Domain 1

Domain & ghost zone
for one stepDomain & ghost zone
for two stepsDomain & ghost zone
for three steps

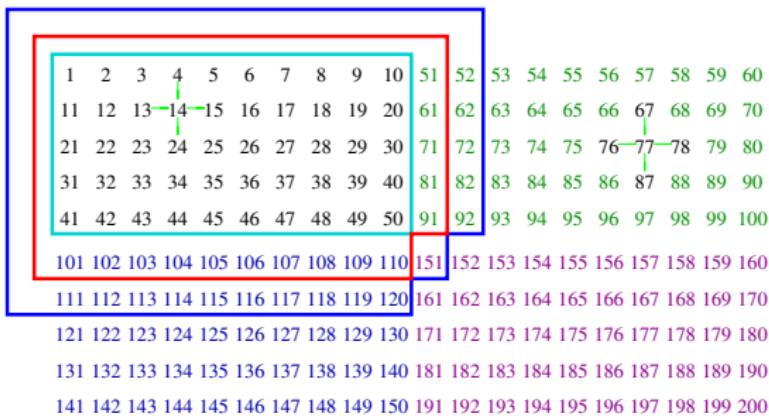


— Domain 1

— Domain & ghost zone
for one step

— Domain & ghost zone
for two steps

— Domain & ghost zone
for three steps

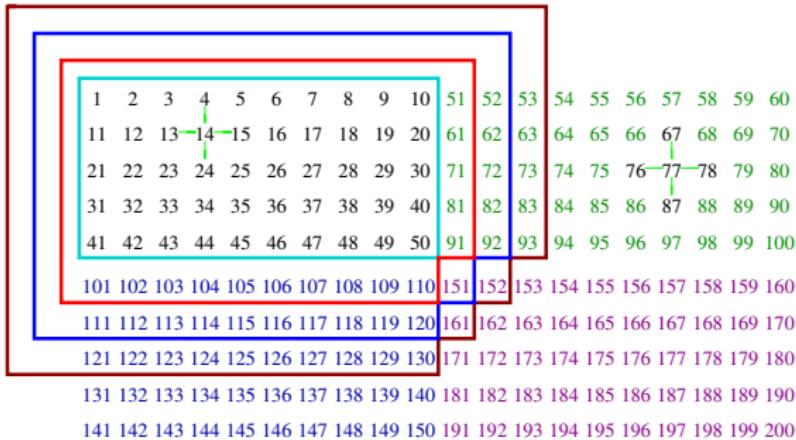


Domain 1

Domain & ghost zone
for one step

Domain & ghost zone
for two steps

Domain & ghost zone
for three steps

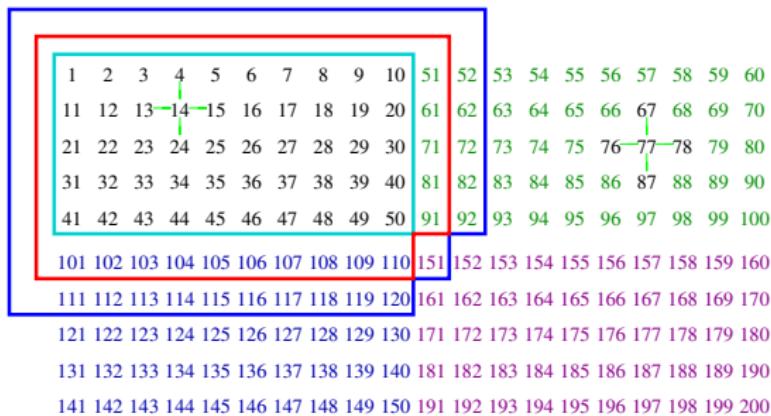


Domain 1

Domain & ghost zone
for one step

Domain & ghost zone
for two steps

Domain & ghost zone
for three steps

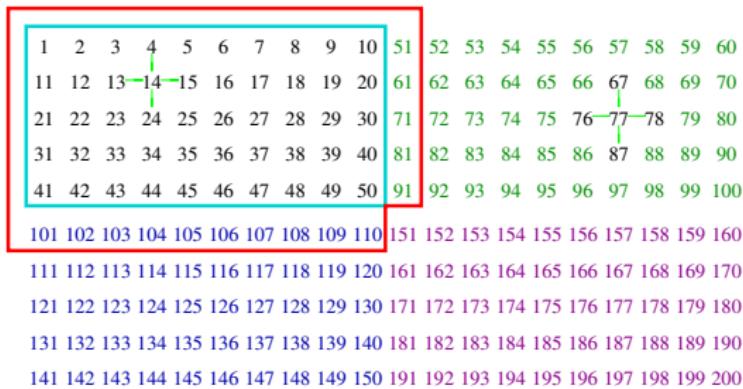


Domain 1

Domain & ghost zone
for one step

Domain & ghost zone
for two steps

Domain & ghost zone
for three steps



Domain 1

Domain & ghost zone
for one step

Domain & ghost zone
for two steps

Domain & ghost zone
for three steps

1	2	3	4	5	6	7	8	9	10	51	52	53	54	55	56	57	58	59	60
11	12	13	14	15	16	17	18	19	20	61	62	63	64	65	66	67	68	69	70
21	22	23	24	25	26	27	28	29	30	71	72	73	74	75	76	77	78	79	80
31	32	33	34	35	36	37	38	39	40	81	82	83	84	85	86	87	88	89	90
41	42	43	44	45	46	47	48	49	50	91	92	93	94	95	96	97	98	99	100

101 102 103 104 105 106 107 108 109 110 151 152 153 154 155 156 157 158 159 160

111 112 113 114 115 116 117 118 119 120 161 162 163 164 165 166 167 168 169 170

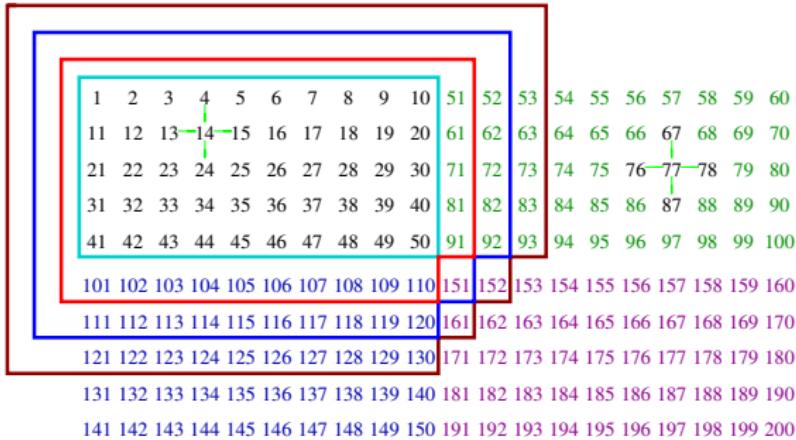
121 122 123 124 125 126 127 128 129 130 171 172 173 174 175 176 177 178 179 180

131 132 133 134 135 136 137 138 139 140 181 182 183 184 185 186 187 188 189 190

141 142 143 144 145 146 147 148 149 150 191 192 193 194 195 196 197 198 199 200

Domain 1

Domain & ghost zone
for one stepDomain & ghost zone
for two stepsDomain & ghost zone
for three steps



Domain 1

Domain & ghost zone
for one step

Domain & ghost zone
for two steps

Domain & ghost zone
for three steps

ILU Preconditioner

- ILU preconditioning techniques based on the incomplete factorization of A

$$A = LU + R$$

$$M = LU$$

ILU Preconditioner

- ILU preconditioning techniques based on the incomplete factorization of A

$$A = LU + R$$

$$M = LU$$

- There are 3 main types of ILU preconditioners

ILU Preconditioner

- ILU preconditioning techniques based on the incomplete factorization of A

$$A = LU + R$$

$$M = LU$$

- There are 3 main types of ILU preconditioners
 - ▶ The zero fill-in ILU (ILU(0))

ILU Preconditioner

- ILU preconditioning techniques based on the incomplete factorization of A

$$A = LU + R$$

$$M = LU$$

- There are 3 main types of ILU preconditioners
 - ▶ The zero fill-in ILU (ILU(0))

$$A = \begin{pmatrix} \times & \times & 0 & 0 & \times & 0 & \times \\ \times & \times & \times & \times & 0 & 0 & 0 \\ \times & 0 & \times & \times & 0 & 0 & \times \\ 0 & 0 & \times & \times & 0 & \times & \times \\ 0 & \times & 0 & 0 & \times & \times & 0 \\ \times & 0 & \times & 0 & 0 & \times & \times \\ 0 & 0 & \times & 0 & \times & 0 & \times \end{pmatrix}$$

ILU Preconditioner

- ILU preconditioning techniques based on the incomplete factorization of A

$$A = LU + R$$

$$M = LU$$

- There are 3 main types of ILU preconditioners
 - ▶ The zero fill-in ILU (ILU(0))
 - ▶ The level of fill ILU (ILU(k))

$$A = \begin{pmatrix} \times & \times & 0 & 0 & \times & 0 & \times \\ \times & \times & \times & \times & 0 & 0 & 0 \\ \times & 0 & \times & \times & 0 & 0 & \times \\ 0 & 0 & \times & \times & 0 & \times & \times \\ 0 & \times & 0 & 0 & \times & \times & 0 \\ \times & 0 & \times & 0 & 0 & \times & \times \\ 0 & 0 & \times & 0 & \times & 0 & \times \end{pmatrix}$$

ILU Preconditioner

- ILU preconditioning techniques based on the incomplete factorization of A

$$A = LU + R$$

$$M = LU$$

- There are 3 main types of ILU preconditioners
 - ▶ The zero fill-in ILU (ILU(0))
 - ▶ The level of fill ILU (ILU(k))
 - ▶ The drop tolerance ILU

$$A = \begin{pmatrix} \times & \times & 0 & 0 & \times & 0 & \times \\ \times & \times & \times & \times & 0 & 0 & 0 \\ \times & 0 & \times & \times & 0 & 0 & \times \\ 0 & 0 & \times & \times & 0 & \times & \times \\ 0 & \times & 0 & 0 & \times & \times & 0 \\ \times & 0 & \times & 0 & 0 & \times & \times \\ 0 & 0 & \times & 0 & \times & 0 & \times \end{pmatrix}$$

Processor i has to
Compute part α_0 of

$$y_1 = (LU)^{-1}Ay_0$$

$$\begin{aligned}y_2 &= ((LU)^{-1}A)^2y_0 \\&= (LU)^{-1}Ay_1\end{aligned}$$

⋮

$$\begin{aligned}y_s &= ((LU)^{-1}A)^sy_0 \\&= (LU)^{-1}Ay_{s-1}\end{aligned}$$

without communicating
with other processors

The s-step Data Dependencies

Processor i has to
Compute part α_0 of

$$y_1 = (LU)^{-1}Ay_0$$

$$y_i = (LU)^{-1}Ay_{i-1}$$

$$\iff \begin{cases} f = Ay_{i-1} \\ Lz = f \\ Uy_i = z \end{cases}$$

$$y_s = (LU)^{-1}Ay_{s-1}$$

without communicating
with other processors

The s-step Data Dependencies

Processor i has to
Compute part α_0 of

$$y_1 = (LU)^{-1}Ay_0$$

$$y_i = (LU)^{-1}Ay_{i-1}$$

$$\iff \begin{cases} f = Ay_{i-1} \\ Lz = f \\ Uy_i = z \end{cases}$$

$$y_s = (LU)^{-1}Ay_{s-1}$$

without communicating
with other processors

s-step Dependencies (A, L, U, s, α_0)

for $j = 1$ to s

Find $\beta_j = R(G(U), \alpha_{j-1})$

Find $\gamma_j = R(G(L), \beta_j)$

Find $\delta_j = Adj(G(A), \gamma_j)$

Set $\alpha_j = \delta_j$

end

ILU Matrix Powers Kernel

ILU Matrix Powers Kernel (A, L, U, s, α_0)

- Processor i calls the algorithm s-step Dependencies
- Processor i fetches the corresponding parts of A, L, U, y_0
- for $j = s$ to 1
 - Compute $f(\gamma_j) = A(\gamma_j, \delta_j)y_{j-k}(\delta_j)$
 - Solve $L(\gamma_j, \gamma_j)z_{j-k+1}(\gamma_j) = f(\gamma_j)$
 - Solve $U(\beta_j, \beta_j)y_{j-k+1}(\beta_j) = z(\beta_j)$
 - Save $y_{j-k+1}(\alpha_0)$, which is the part that Processor i has to compute
 - Set $y_{j-k} = y_{j-k+1}$

s-step Dependencies

(A, L, U, s, α_0)

for $j = 1$ to s

Find $\beta_j = R(G(U), \alpha_{j-1})$

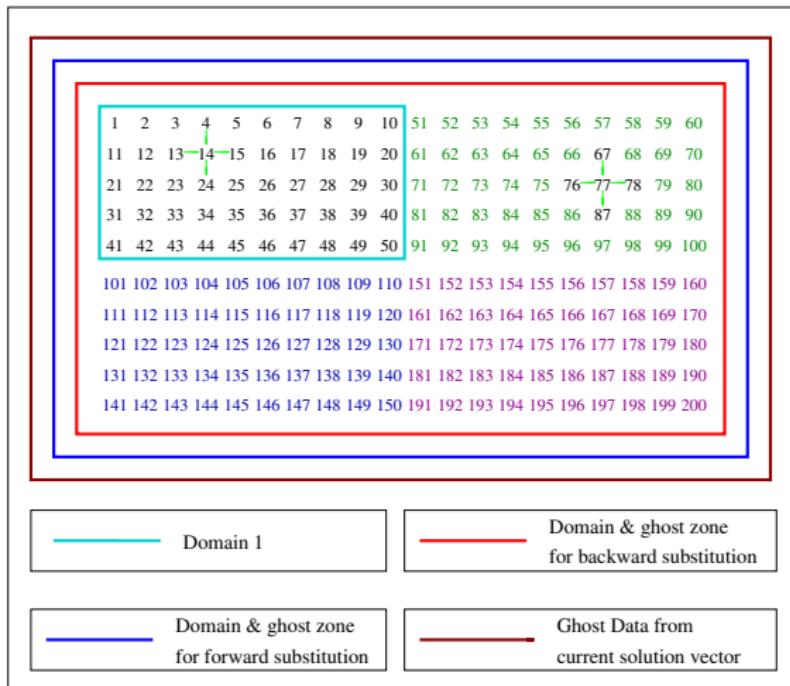
Find $\gamma_j = R(G(L), \beta_j)$

Find $\delta_j = Adj(G(A), \gamma_j)$

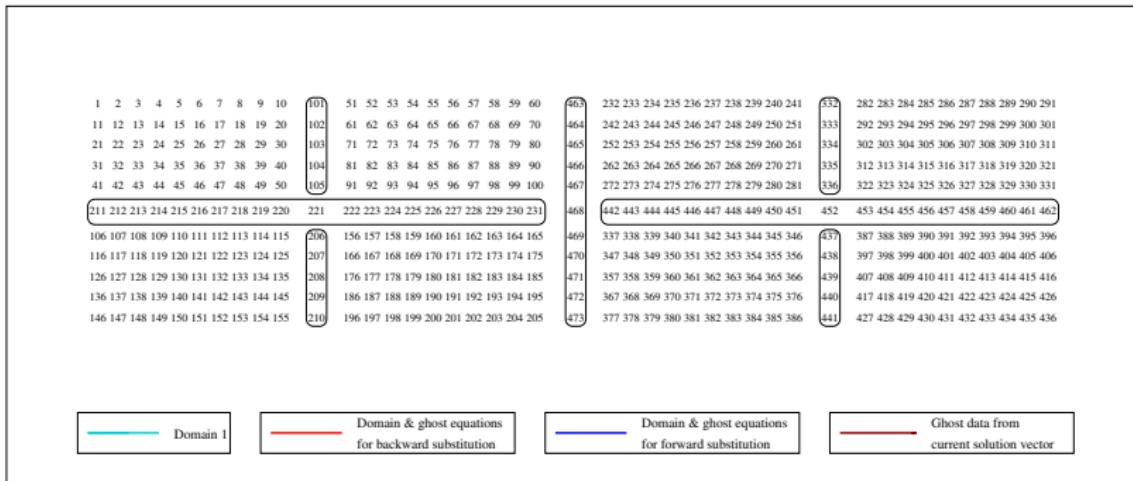
Set $\alpha_j = \delta_j$

end

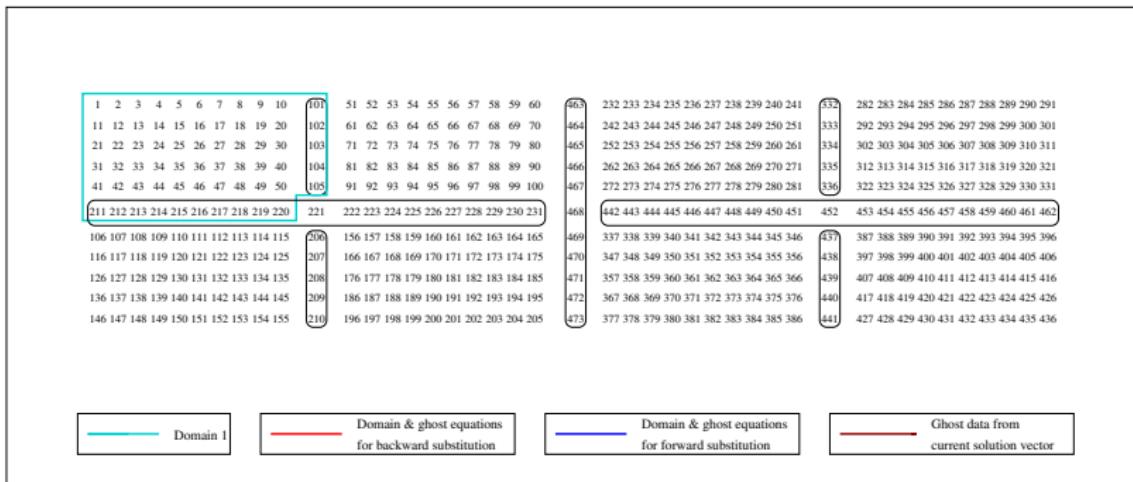
ILU0 Matrix Powers Kernel for $s = 1$



Nested Dissection ILU0 Matrix Powers Kernel for $s = 1$



Nested Dissection ILU0 Matrix Powers Kernel for $s = 1$



Nested Dissection ILU0 Matrix Powers Kernel for $s = 1$

Domain										Domain & ghost equations for backward substitution										Domain & ghost equations for forward substitution										Ghost data from current solution vector												
1	2	3	4	5	6	7	8	9	10	101	51	52	53	54	55	56	57	58	59	60	667	232	233	234	235	236	237	238	239	240	241	332	282	283	284	285	286	287	288	289	290	291
11	12	13	14	15	16	17	18	19	20	102	61	62	63	64	65	66	67	68	69	70	464	242	243	244	245	246	247	248	249	250	251	333	292	293	294	295	296	297	298	299	300	301
21	22	23	24	25	26	27	28	29	30	103	71	72	73	74	75	76	77	78	79	80	465	252	253	254	255	256	257	258	259	260	261	334	302	303	304	305	306	307	308	309	310	311
31	32	33	34	35	36	37	38	39	40	104	81	82	83	84	85	86	87	88	89	90	466	262	263	264	265	266	267	268	269	270	271	335	312	313	314	315	316	317	318	319	320	321
41	42	43	44	45	46	47	48	49	50	105	91	92	93	94	95	96	97	98	99	100	467	272	273	274	275	276	277	278	279	280	281	336	322	323	324	325	326	327	328	329	330	331
211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	468	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462
106	107	108	109	110	111	112	113	114	115	209	156	157	158	159	160	161	162	163	164	165	469	337	338	339	340	341	342	343	344	345	346	437	387	388	389	390	391	392	393	394	395	396
116	117	118	119	120	121	122	123	124	125	207	166	167	168	169	170	171	172	173	174	175	470	347	348	349	350	351	352	353	354	355	356	438	397	398	399	400	401	402	403	404	405	406
126	127	128	129	130	131	132	133	134	135	208	176	177	178	179	180	181	182	183	184	185	471	357	358	359	360	361	362	363	364	365	366	439	407	408	409	410	411	412	413	414	415	416
136	137	138	139	140	141	142	143	144	145	209	186	187	188	189	190	191	192	193	194	195	472	367	368	369	370	371	372	373	374	375	376	440	417	418	419	420	421	422	423	424	425	426
146	147	148	149	150	151	152	153	154	155	210	196	197	198	199	200	201	202	203	204	205	473	377	378	379	380	381	382	383	384	385	386	441	427	428	429	430	431	432	433	434	435	436

Nested Dissection ILU0 Matrix Powers Kernel for $s = 1$

1 2 3 4 5 6 7 8 9 10	101	51 52 53 54 55 56 57 58 59 60	667	232 233 234 235 236 237 238 239 240 241	332	282 283 284 285 286 287 288 289 290 291
11 12 13 14 15 16 17 18 19 20	102	61 62 63 64 65 66 67 68 69 70	464	242 243 244 245 246 247 248 249 250 251	333	292 293 294 295 296 297 298 299 300 301
21 22 23 24 25 26 27 28 29 30	103	71 72 73 74 75 76 77 78 79 80	465	252 253 254 255 256 257 258 259 260 261	334	302 303 304 305 306 307 308 309 310 311
31 32 33 34 35 36 37 38 39 40	104	81 82 83 84 85 86 87 88 89 90	466	262 263 264 265 266 267 268 269 270 271	335	312 313 314 315 316 317 318 319 320 321
41 42 43 44 45 46 47 48 49 50	105	91 92 93 94 95 96 97 98 99 100	467	272 273 274 275 276 277 278 279 280 281	336	322 323 324 325 326 327 328 329 330 331
211 212 213 214 215 216 217 218 219 220	221	222 223 224 225 226 227 228 229 230 231	468	442 443 444 445 446 447 448 449 450 451	452	453 454 455 456 457 458 459 460 461 462
106 107 108 109 110 111 112 113 114 115	209	156 157 158 159 160 161 162 163 164 165	469	337 338 339 340 341 342 343 344 345 346	437	387 388 389 390 391 392 393 394 395 396
116 117 118 119 120 121 122 123 124 125	207	166 167 168 169 170 171 172 173 174 175	470	347 348 349 350 351 352 353 354 355 356	438	397 398 399 400 401 402 403 404 405 406
126 127 128 129 130 131 132 133 134 135	208	176 177 178 179 180 181 182 183 184 185	471	357 358 359 360 361 362 363 364 365 366	439	407 408 409 410 411 412 413 414 415 416
136 137 138 139 140 141 142 143 144 145	209	186 187 188 189 190 191 192 193 194 195	472	367 368 369 370 371 372 373 374 375 376	440	417 418 419 420 421 422 423 424 425 426
146 147 148 149 150 151 152 153 154 155	210	196 197 198 199 200 201 202 203 204 205	473	377 378 379 380 381 382 383 384 385 386	441	427 428 429 430 431 432 433 434 435 436

Domain 1

Domain & ghost equations
for backward substitution

Domain & ghost equations
for forward substitution

Ghost data from
current solution vector

Nested Dissection ILU0 Matrix Powers Kernel for $s = 1$

1 2 3 4 5 6 7 8 9 10	101	51 52 53 54 55 56 57 58 59 60	463	232 233 234 235 236 237 238 239 240 241	332	282 283 284 285 286 287 288 289 290 291
11 12 13 14 15 16 17 18 19 20	102	61 62 63 64 65 66 67 68 69 70	464	242 243 244 245 246 247 248 249 250 251	333	292 293 294 295 296 297 298 299 300 301
21 22 23 24 25 26 27 28 29 30	103	71 72 73 74 75 76 77 78 79 80	465	252 253 254 255 256 257 258 259 260 261	334	302 303 304 305 306 307 308 309 310 311
31 32 33 34 35 36 37 38 39 40	104	81 82 83 84 85 86 87 88 89 90	466	262 263 264 265 266 267 268 269 270 271	335	312 313 314 315 316 317 318 319 320 321
41 42 43 44 45 46 47 48 49 50	105	91 92 93 94 95 96 97 98 99 100	467	272 273 274 275 276 277 278 279 280 281	336	322 323 324 325 326 327 328 329 330 331
211 212 213 214 215 216 217 218 219 220	221	222 223 224 225 226 227 228 229 230 231	468	442 443 444 445 446 447 448 449 450 451	452	453 454 455 456 457 458 459 460 461 462
106 107 108 109 110 111 112 113 114 115	209	156 157 158 159 160 161 162 163 164 165	469	337 338 339 340 341 342 343 344 345 346	437	387 388 389 390 391 392 393 394 395 396
116 117 118 119 120 121 122 123 124 125	207	166 167 168 169 170 171 172 173 174 175	470	347 348 349 350 351 352 353 354 355 356	438	397 398 399 400 401 402 403 404 405 406
126 127 128 129 130 131 132 133 134 135	208	176 177 178 179 180 181 182 183 184 185	471	357 358 359 360 361 362 363 364 365 366	439	407 408 409 410 411 412 413 414 415 416
136 137 138 139 140 141 142 143 144 145	209	186 187 188 189 190 191 192 193 194 195	472	367 368 369 370 371 372 373 374 375 376	440	417 418 419 420 421 422 423 424 425 426
146 147 148 149 150 151 152 153 154 155	210	196 197 198 199 200 201 202 203 204 205	473	377 378 379 380 381 382 383 384 385 386	441	427 428 429 430 431 432 433 434 435 436

Domain 1

Domain & ghost equations
for backward substitution

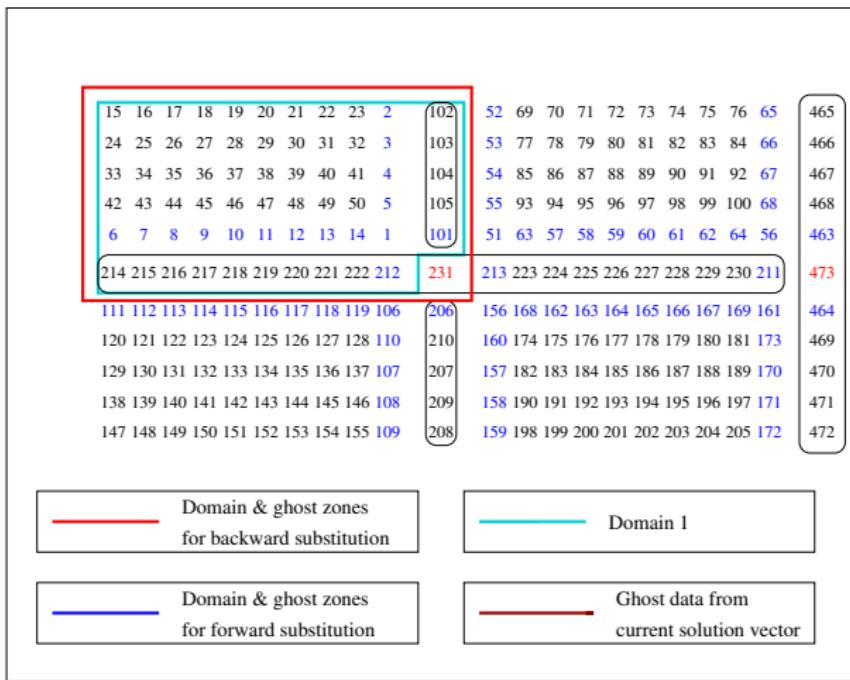
Domain & ghost equations
for forward substitution

Ghost data from
current solution vector

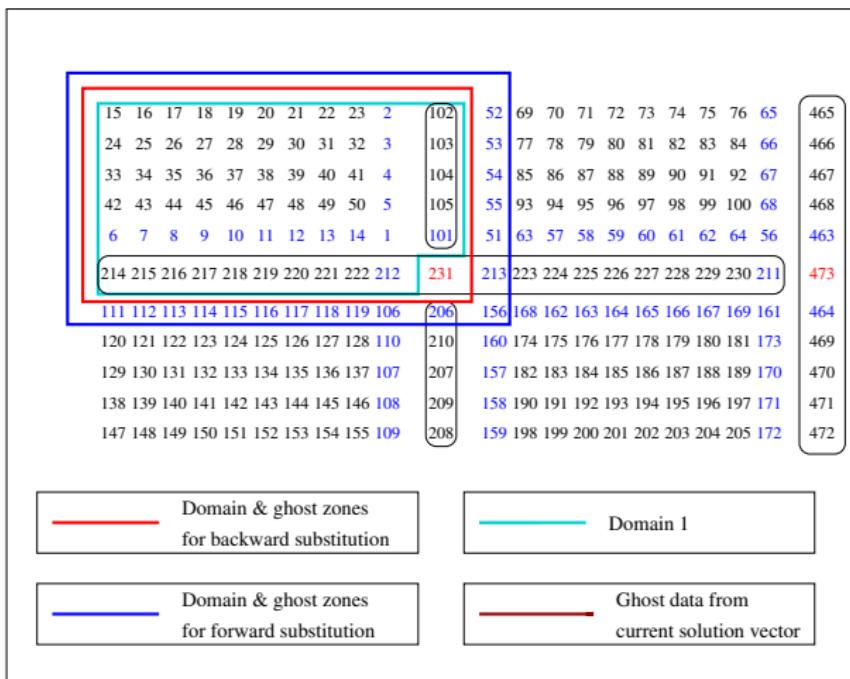
CR-ILU0 reordering



CR-ILU0 reordering



CR-ILU0 reordering



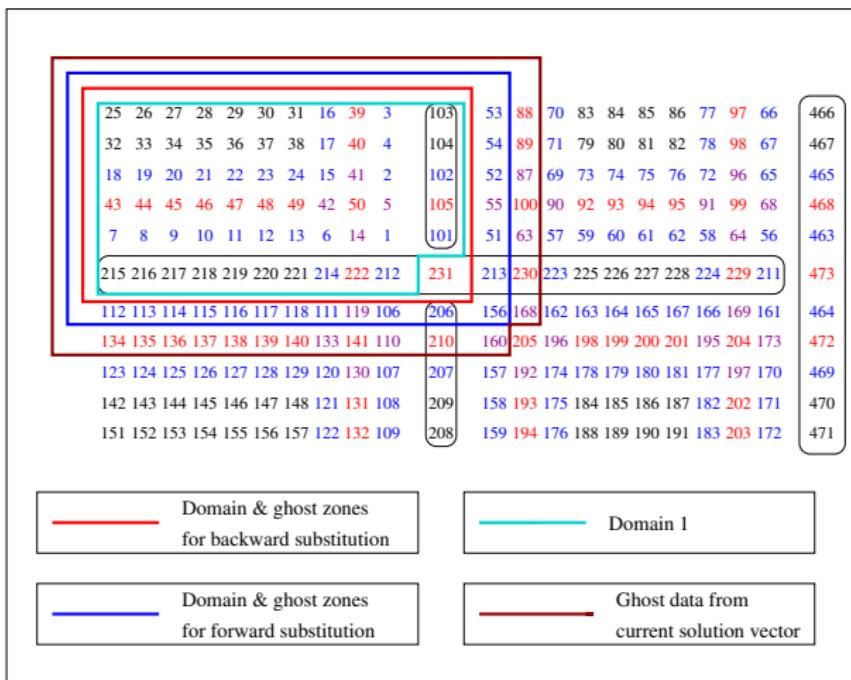
CR-ILU0 reordering



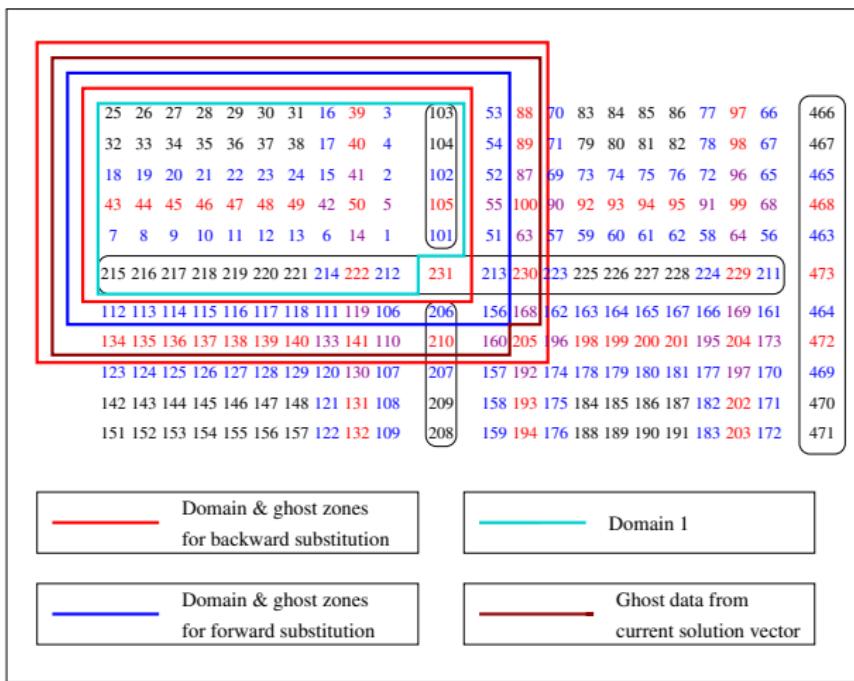
CA-ILU0(s) reordering



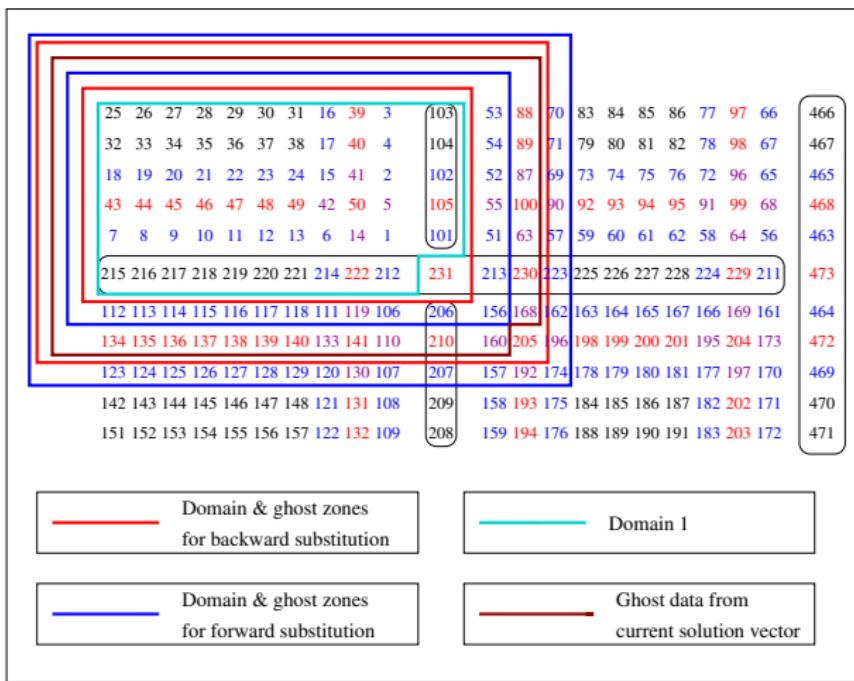
CA-ILU0(s) reordering



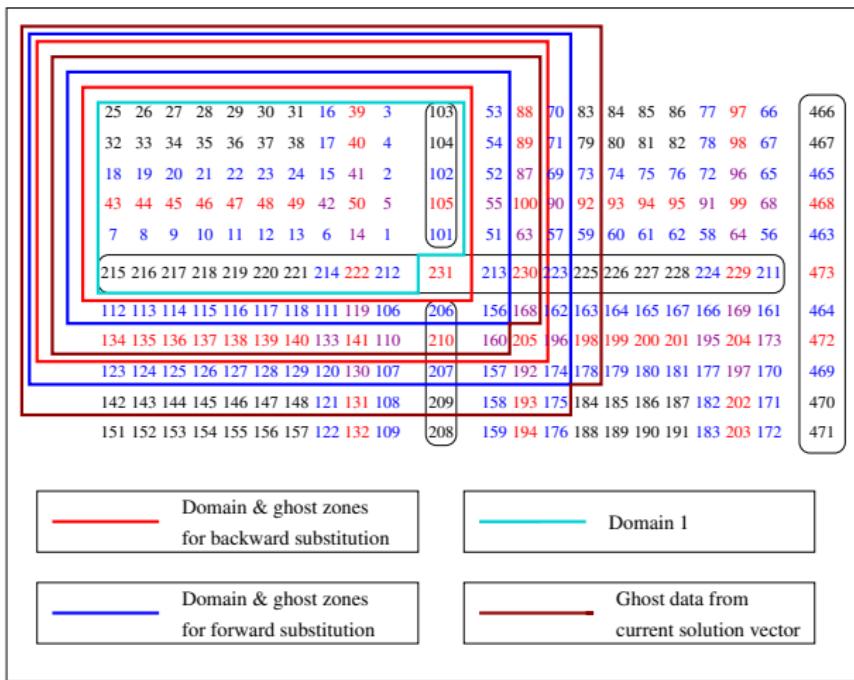
CA-ILU0(s) reordering



CA-ILU0(s) reordering



CA-ILU0(s) reordering



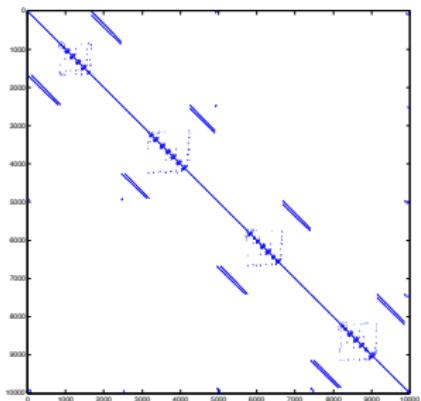
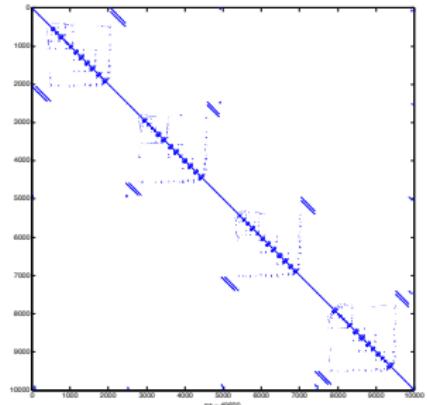
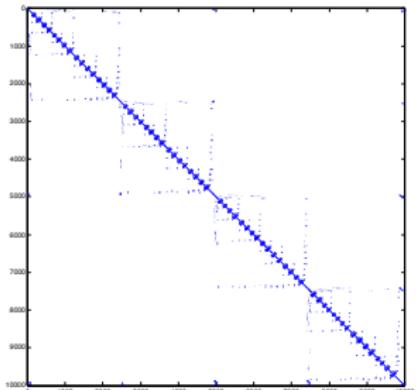
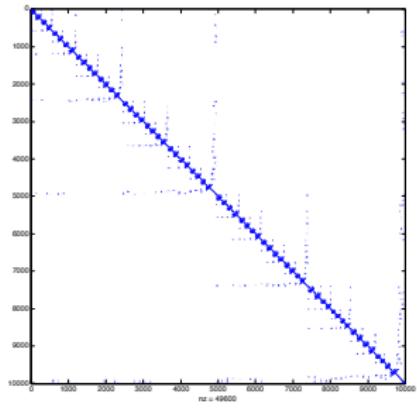


Figure:
Comparison of
the sparsity
patterns of the
ND, CR-ILU0,
CA-ILU0(5)
and
CA-ILU0(10)
reordering
matrix
matvf2DNH100100
of size
 $10^4 \times 10^4$ with
 $P=4$

The complexity of the CA-ILU0(s) reordering

- ▶ The complexity of CA-ILU0(s) is defined as the number of times the vertices and the edges in the graph of A are visited/read in order to perform the reordering.

The complexity of the CA-ILU0(s) reordering

- ▶ The complexity of CA-ILU0(s) is defined as the number of times the vertices and the edges in the graph of A are visited/read in order to perform the reordering.
- ▶ CA-ILU0(s) reordering is done in parallel on P processors,

The complexity of the CA-ILU0(s) reordering

- ▶ The complexity of CA-ILU0(s) is defined as the number of times the vertices and the edges in the graph of A are visited/read in order to perform the reordering.
- ▶ CA-ILU0(s) reordering is done in parallel on P processors, then the parallel complexity is upper bounded by

$$2|D_t(l_{max})| + (3\log(P) - 2)|S_{max}(m_{max})|$$

where $D_t(l_{max})$ is the largest subdomain and $S_{max}(m_{max})$ is the largest separator.

The complexity of the CA-ILU0(s) reordering

- ▶ The complexity of CA-ILU0(s) is defined as the number of times the vertices and the edges in the graph of A are visited/read in order to perform the reordering.
- ▶ CA-ILU0(s) reordering is done in parallel on P processors, then the parallel complexity is upper bounded by

$$2|D_t(l_{max})| + (3\log(P) - 2)|S_{max}(m_{max})|$$

where $D_t(l_{max})$ is the largest subdomain and $S_{max}(m_{max})$ is the largest separator.

- ▶ Thus, our algorithm is of linear complexity.

CA-ILU0 Preconditioner

1. Partitioning the graph of A to obtain P subdomains

CA-ILU0 Preconditioner

1. Partitioning the graph of A to obtain P subdomains
2. Each processor is assigned a subdomain and reorders its subdomains in parallel, using the CA-ILU0(s) reordering

CA-ILU0 Preconditioner

1. Partitioning the graph of A to obtain P subdomains
2. Each processor is assigned a subdomain and reorders its subdomains in parallel, using the CA-ILU0(s) reordering
3. Then finds the redundant/ghost data that it needs

CA-ILU0 Preconditioner

1. Partitioning the graph of A to obtain P subdomains
2. Each processor is assigned a subdomain and reorders its subdomains in parallel, using the CA-ILU0(s) reordering
3. Then finds the redundant/ghost data that it needs
4. Each processor i fetches its corresponding $A(\gamma_s, :)$ and $y_0(\delta_s)$

CA-ILU0 Preconditioner

1. Partitioning the graph of A to obtain P subdomains
2. Each processor is assigned a subdomain and reorders its subdomains in parallel, using the CA-ILU0(s) reordering
3. Then finds the redundant/ghost data that it needs
4. Each processor i fetches its corresponding $A(\gamma_s, :)$ and $y_0(\delta_s)$
5. Each processor i performs the CA-ILU(0) factorization of $A(\gamma_s, :)$ to obtain the corresponding $L(\gamma_s, :)$ and $U(\gamma_s, :)$ matrices

CA-ILU0 Preconditioner

1. Partitioning the graph of A to obtain P subdomains
2. Each processor is assigned a subdomain and reorders its subdomains in parallel, using the CA-ILU0(s) reordering
3. Then finds the redundant/ghost data that it needs
4. Each processor i fetches its corresponding $A(\gamma_s, :)$ and $y_0(\delta_s)$
5. Each processor i performs the CA-ILU(0) factorization of $A(\gamma_s, :)$ to obtain the corresponding $L(\gamma_s, :)$ and $U(\gamma_s, :)$ matrices
6. Each processor i performs the s multiplications of the form $y_{i+1} = [L(\gamma_{s-i}, \gamma_{s-i})U(\beta_{s-i}, \beta_{s-i})]^{-1}A(\gamma_{s-i}, \delta_{s-i})y_i(\delta_{s-i})$ using the ILU Matrix Powers Kernel

CA-ILU0 Preconditioner

1. Partitioning the graph of A to obtain P subdomains
2. Each processor is assigned a subdomain and reorders its subdomains in parallel, using the CA-ILU0(s) reordering
3. Then finds the redundant/ghost data that it needs
4. Each processor i fetches its corresponding $A(\gamma_s, :)$ and $y_0(\delta_s)$
5. Each processor i performs the CA-ILU(0) factorization of $A(\gamma_s, :)$ to obtain the corresponding $L(\gamma_s, :)$ and $U(\gamma_s, :)$ matrices
6. Each processor i performs the s multiplications of the form $y_{i+1} = [L(\gamma_{s-i}, \gamma_{s-i})U(\beta_{s-i}, \beta_{s-i})]^{-1}A(\gamma_{s-i}, \delta_{s-i})y_i(\delta_{s-i})$ using the ILU Matrix Powers Kernel
7. The processors communicate to fetch the needed data to proceed with the CA-GMRES algorithm

CA-ILU0 Preconditioner

1. Partitioning the graph of A to obtain P subdomains
2. Each processor is assigned a subdomain and reorders its subdomains in parallel, using the CA-ILU0(s) reordering
3. Then finds the redundant/ghost data that it needs
4. Each processor i fetches its corresponding $A(\gamma_s, :)$ and $y_0(\delta_s)$
5. Each processor i performs the CA-ILU(0) factorization of $A(\gamma_s, :)$ to obtain the corresponding $L(\gamma_s, :)$ and $U(\gamma_s, :)$ matrices
6. Each processor i performs the s multiplications of the form $y_{i+1} = [L(\gamma_{s-i}, \gamma_{s-i})U(\beta_{s-i}, \beta_{s-i})]^{-1}A(\gamma_{s-i}, \delta_{s-i})y_i(\delta_{s-i})$ using the ILU Matrix Powers Kernel
7. The processors communicate to fetch the needed data to proceed with the CA-GMRES algorithm
8. If the method hasn't converged, then each processor fetches $y_s(\delta_s)$ and performs step 6 and on.

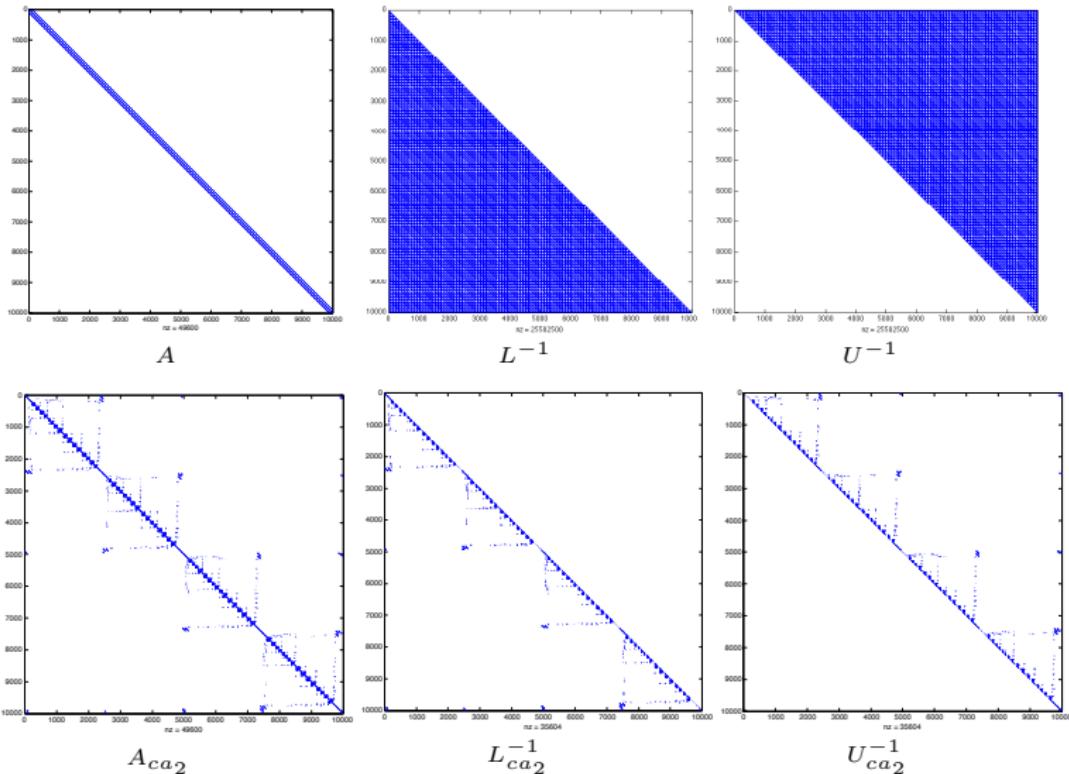
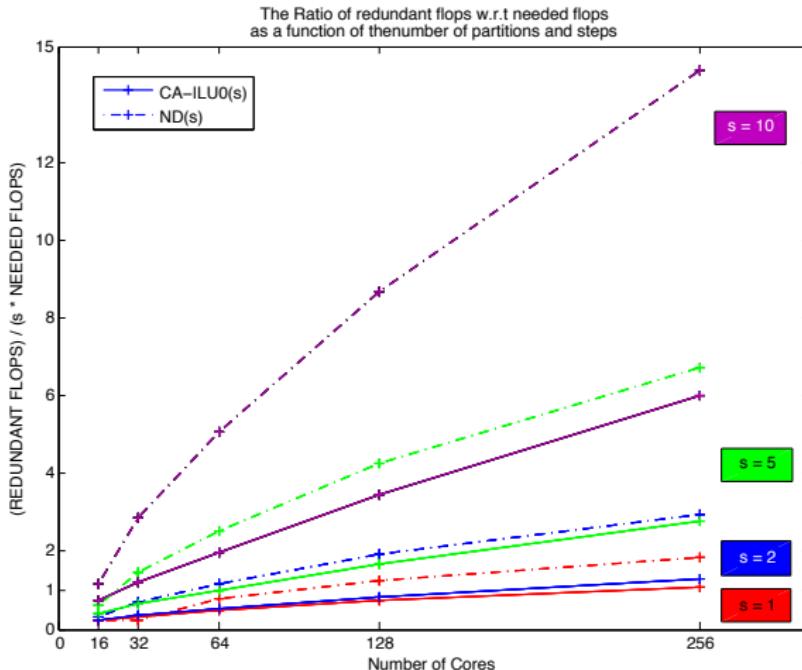
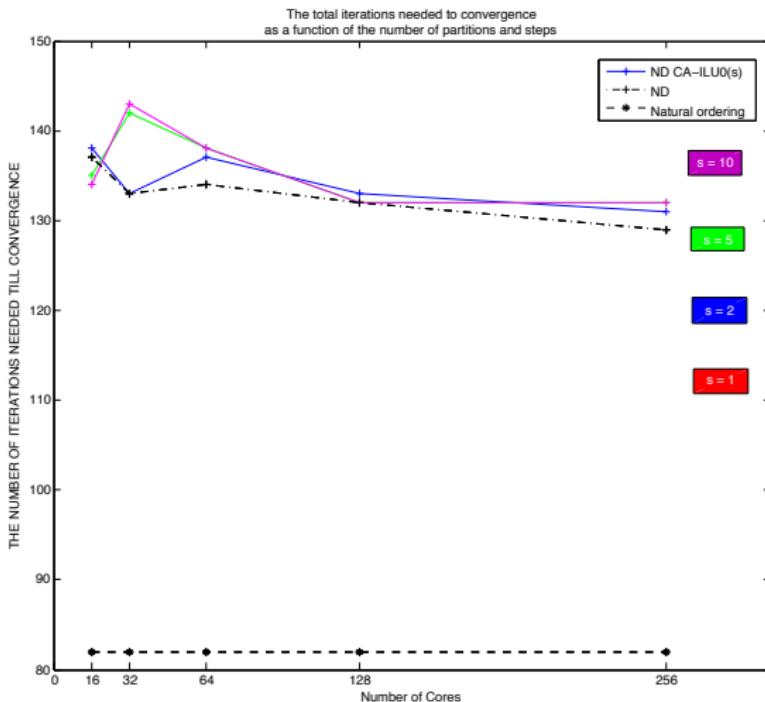


Figure: The fill-ins in the ILU(0) factorization of a 2D matrix A and its CA-ILU0(2) reordered version A_{ca}

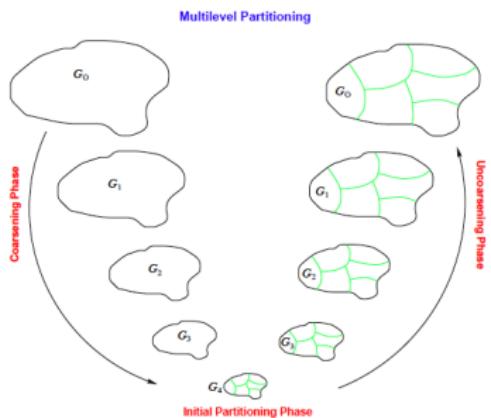
Redundant Flops in ILU-Matrix Powers Kernel



The effect of reordering A on GMRES's Convergence

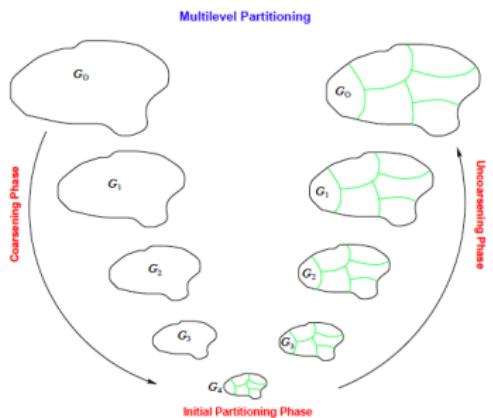


k-way Partitioning



Source: METIS 5.0 Manual

k-way Partitioning

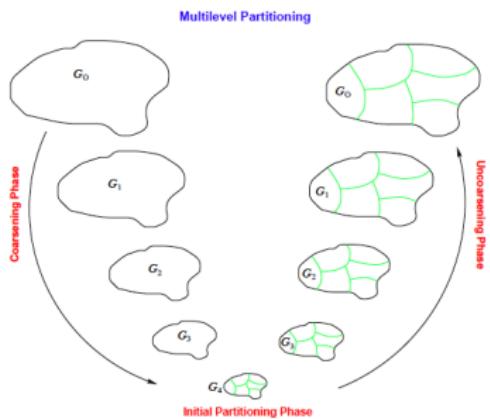


Source: METIS 5.0 Manual

The three phases of multilevel k-way graph partitioning:

- ▶ The coarsening phase: the size of the graph is successively decreased
- ▶ The initial partitioning phase: a k-way partitioning is computed

k-way Partitioning



The three phases of multilevel k-way graph partitioning:

- ▶ The coarsening phase: the size of the graph is successively decreased
- ▶ The initial partitioning phase: a k-way partitioning is computed
- ▶ The multilevel refinement (or uncoarsening) phase: the partitioning is successively refined as it is projected to the larger graphs.

Source: METIS 5.0 Manual

CR-ILU0 reordering

13 14 15 16 17 18 19 20 2 47	97 52 63 64 65 66 67 68 69 70
21 22 23 24 25 26 27 28 3 48	98 53 71 72 73 74 75 76 77 78
29 30 31 32 33 34 35 36 4 49	99 54 79 80 81 82 83 84 85 86
5 6 7 8 9 10 11 12 1 46	96 51 55 56 57 58 59 60 61 62
38 39 40 41 42 43 44 45 37 50	100 87 88 89 90 91 92 93 94 95
138 139 140 141 142 143 144 145 137 150	200 187 188 189 190 191 192 193 194 195
105 106 107 108 109 110 111 112 101 146	196 151 155 156 157 158 159 160 161 162
113 114 115 116 117 118 119 120 102 147	197 152 163 164 165 166 167 168 169 170
121 122 123 124 125 126 127 128 103 148	198 153 171 172 173 174 175 176 177 178
129 130 131 132 133 134 135 136 104 149	199 154 179 180 181 182 183 184 185 186

— Domain 1

— Domain & ghost zone
for backward substitution

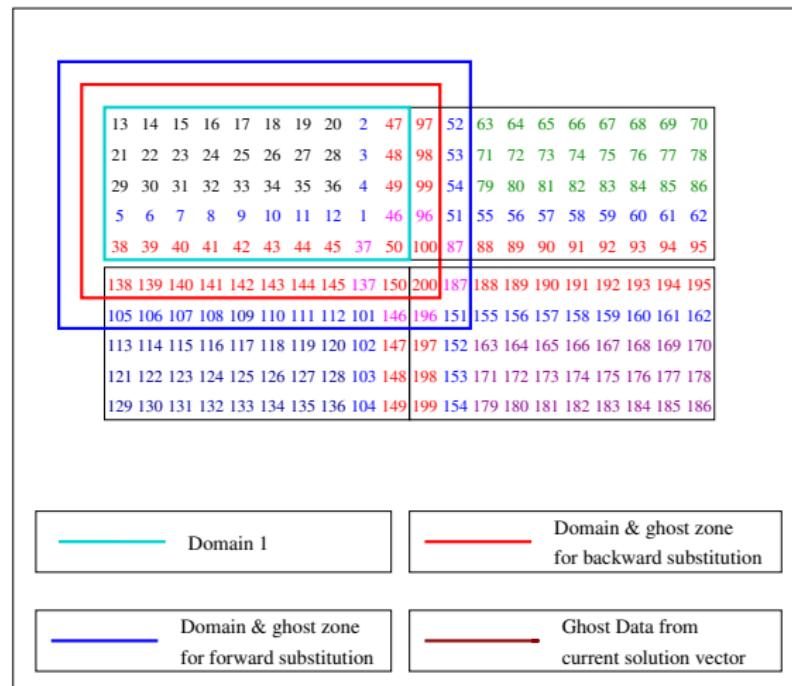
— Domain & ghost zone
for forward substitution

— Ghost Data from
current solution vector

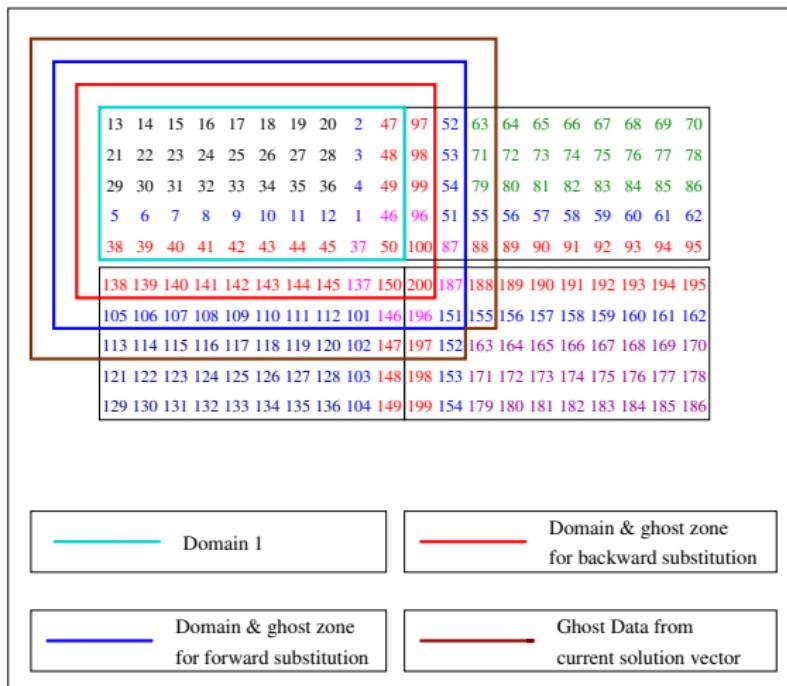
CR-ILU0 reordering



CR-ILU0 reordering



CR-ILU0 reordering



CA-ILU0(s) reordering

21 22 23 24 25 26 14 35 3 48	98 53 78 64 71 72 73 74 75 76
15 16 17 18 19 20 13 34 2 47	97 52 77 63 65 66 67 68 69 70
28 29 30 31 32 33 27 36 4 49	99 54 86 79 80 81 82 83 84 85
6 7 8 9 10 11 5 12 1 46	96 51 62 55 56 57 58 59 60 61
39 40 41 42 43 44 38 45 37 50	100 87 95 88 89 90 91 92 93 94
139 140 141 142 143 144 138 145 137 150	200 187 195 188 189 190 191 192 193 194
106 107 108 109 110 111 105 112 101 146	196 151 162 155 156 157 158 159 160 161
128 129 130 131 132 133 127 136 104 149	199 154 186 179 185 184 183 182 181 180
115 116 117 118 119 120 113 134 102 147	197 152 177 163 165 166 167 168 169 170
121 122 123 124 125 126 114 135 103 148	198 153 178 164 171 172 173 174 175 176

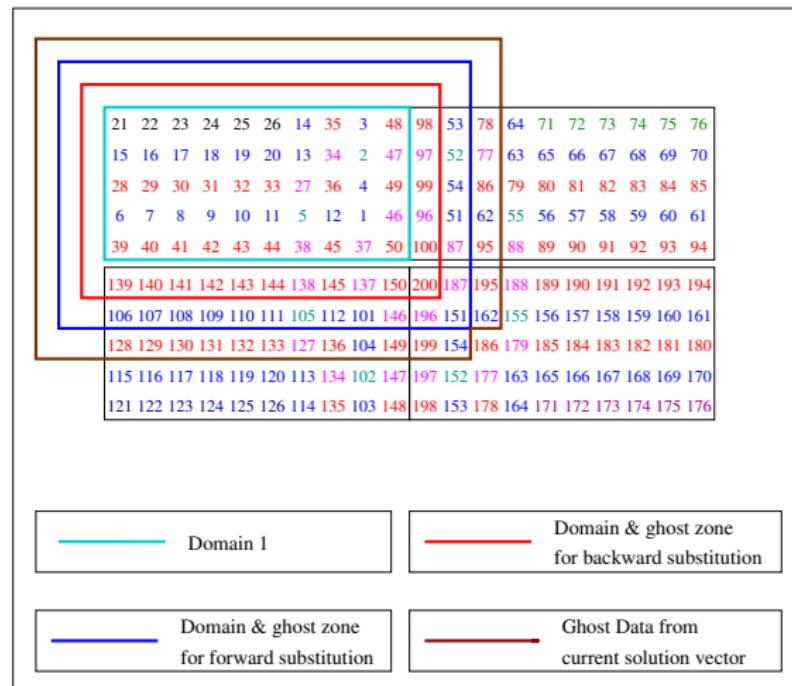
— Domain 1

— Domain & ghost zone
for backward substitution

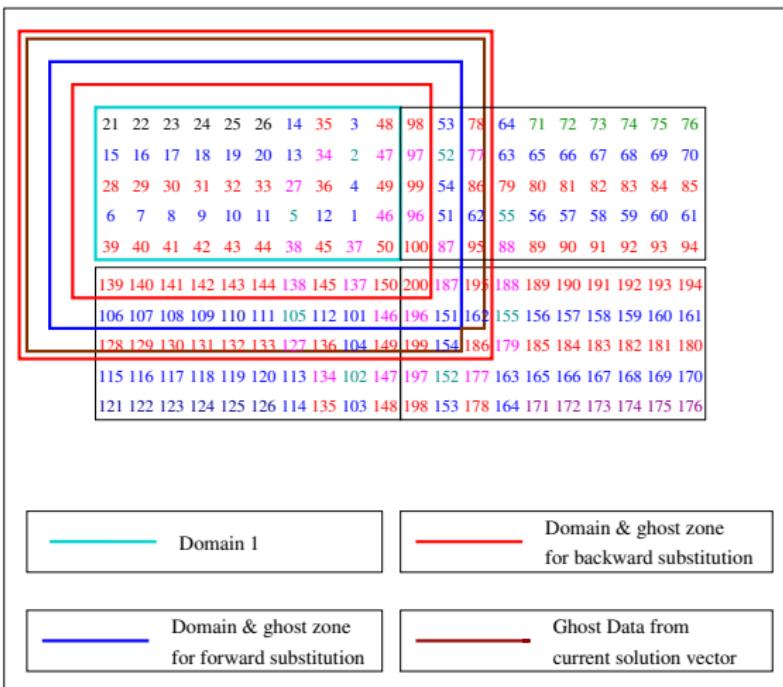
— Domain & ghost zone
for forward substitution

— Ghost Data from
current solution vector

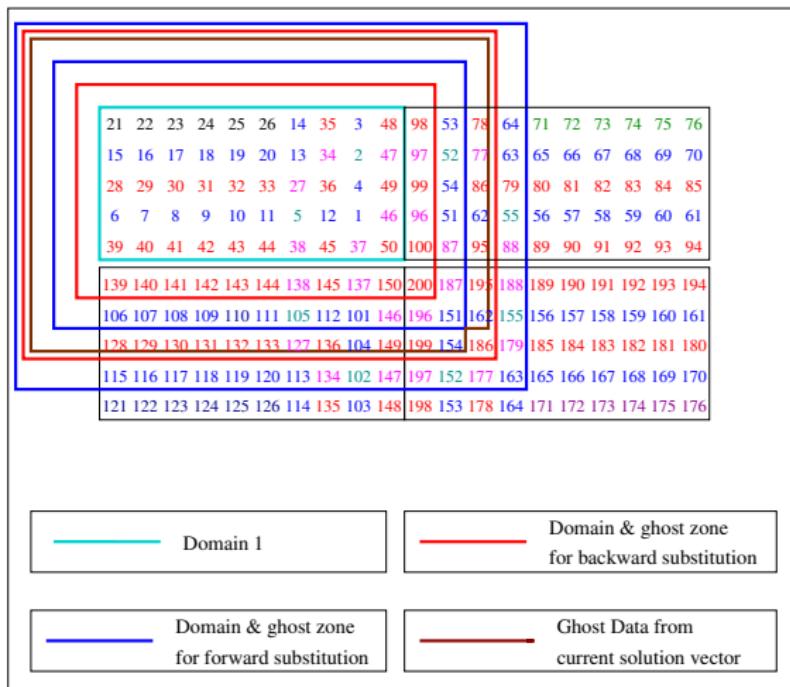
CA-ILU0(s) reordering



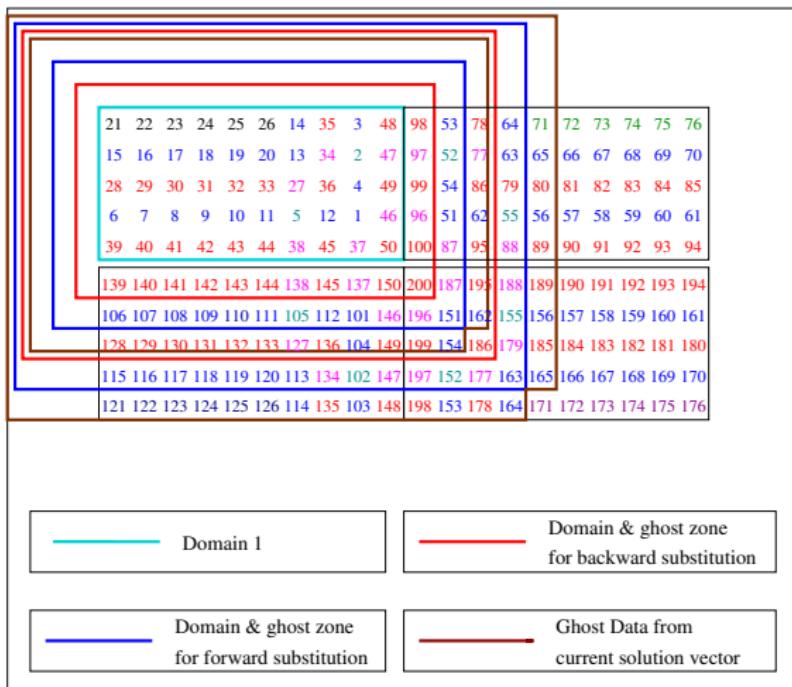
CA-ILU0(s) reordering



CA-ILU0(s) reordering



CA-ILU0(s) reordering



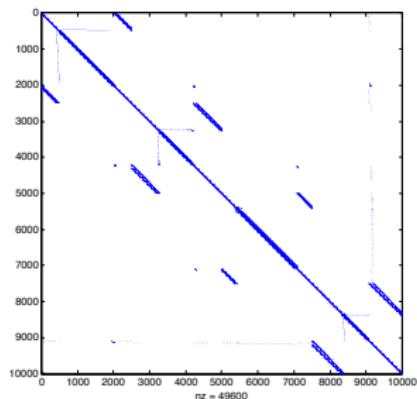
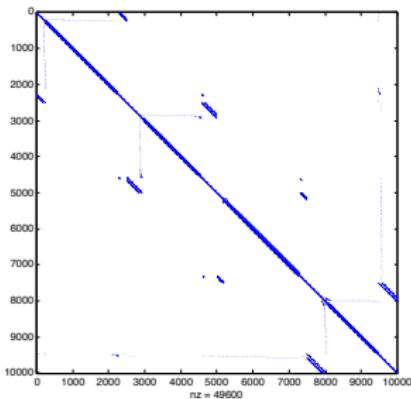
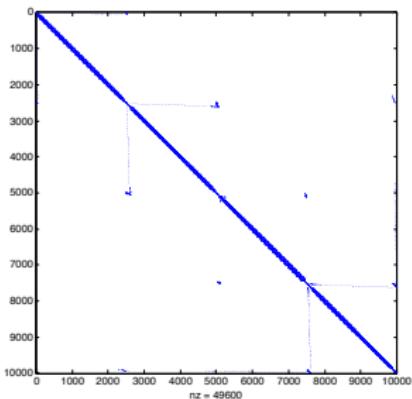
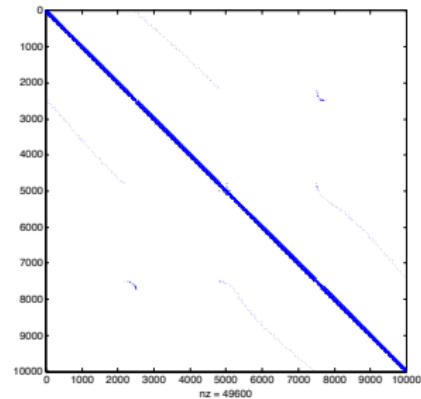
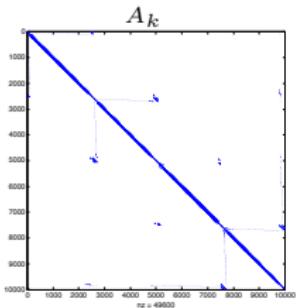
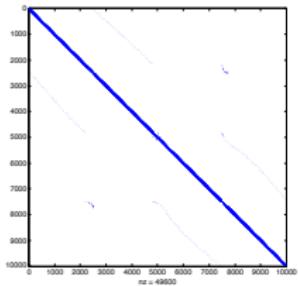
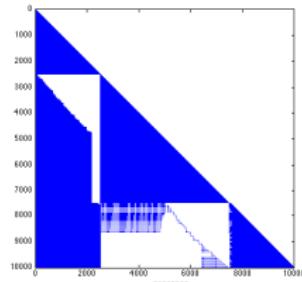


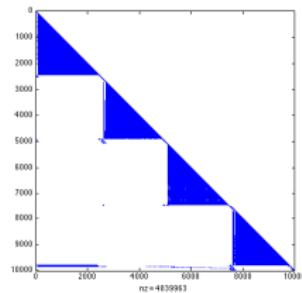
Figure:
Comparison of
the sparsity
patterns of the
k-way,
CR-ILU0,
CA-ILU0(5)
and
CA-ILU0(10)
reordered
matrix
matvf2DNH100100
of size
 $10^4 \times 10^4$ with
 $P=4$



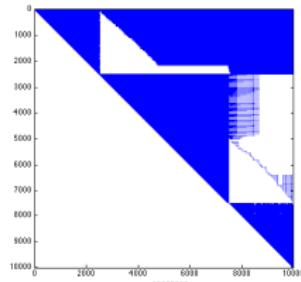
A_{ca2}



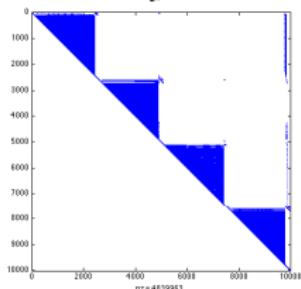
L_t^{-1}



L_{ca2}^{-1}



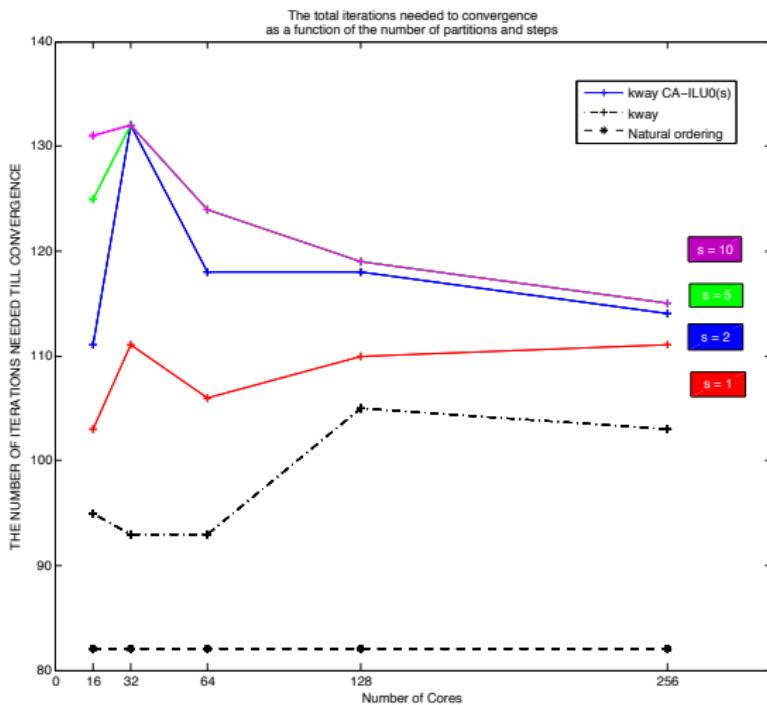
U_t^{-1}



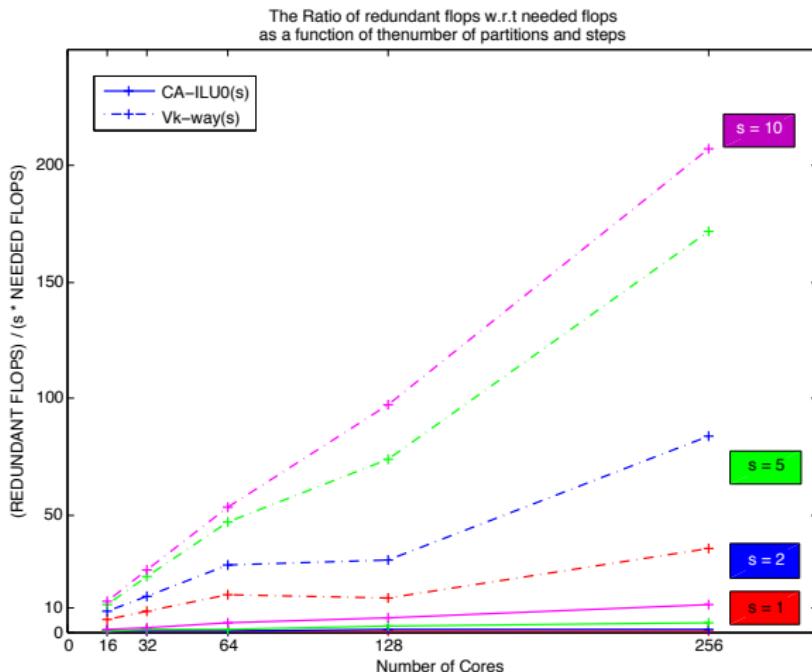
U_{ca2}^{-1}

Figure: The fill-ins in the ILU(0) factorization of a k-way reordered matrix A and its CA-ILU0(2) reordered version A_{ca}

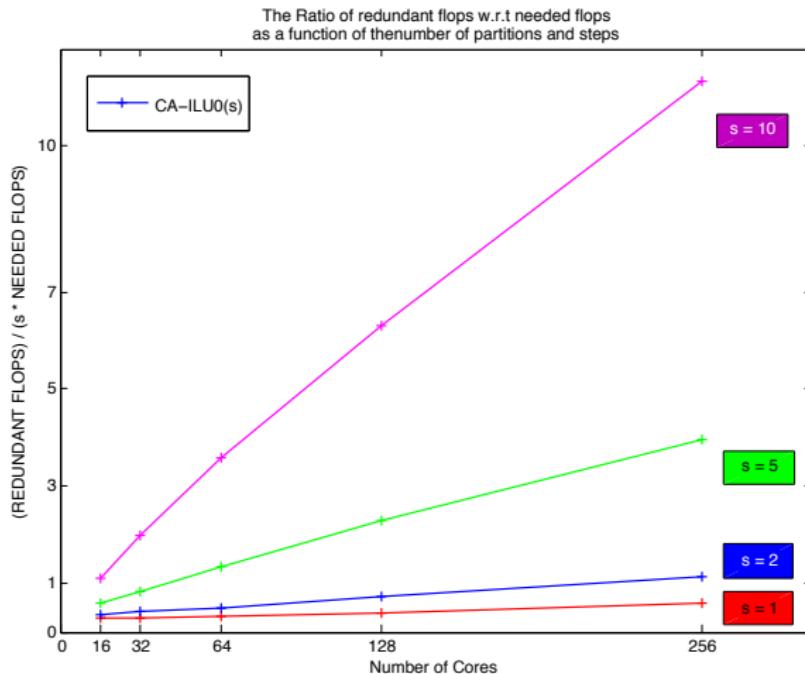
The effect of reordering A on GMRES's Convergence



Redundant Flops in ILU-Matrix Powers Kernel



Redundant Flops in ILU-Matrix Powers Kernel



Conclusion

In this talk we have presented:

- ▶ **ILU Matrix Powers kernel** which is an adaptation of the Matrix Powers Kernel to the case of ILU-preconditioned systems

Conclusion

In this talk we have presented:

- ▶ **ILU Matrix Powers kernel** which is an adaptation of the Matrix Powers Kernel to the case of ILU-preconditioned systems
- ▶ **CR-ILU0 reordering** of the matrix A used for reducing communication in ILU0 preconditioned GMRES and ILU0 preconditioned s-step GMRES.

Conclusion

In this talk we have presented:

- ▶ **ILU Matrix Powers kernel** which is an adaptation of the Matrix Powers Kernel to the case of ILU-preconditioned systems
- ▶ **CR-ILU0 reordering** of the matrix A used for reducing communication in ILU0 preconditioned GMRES and ILU0 preconditioned s-step GMRES.
- ▶ **CA-ILU0(s) reordering** of the matrix A used for avoiding communication in ILU0 preconditioned s-step GMRES and ILU0 preconditioned CA-GMRES.

Conclusion

In this talk we have presented:

- ▶ **ILU Matrix Powers kernel** which is an adaptation of the Matrix Powers Kernel to the case of ILU-preconditioned systems
- ▶ **CR-ILU0 reordering** of the matrix A used for reducing communication in ILU0 preconditioned GMRES and ILU0 preconditioned s-step GMRES.
- ▶ **CA-ILU0(s) reordering** of the matrix A used for avoiding communication in ILU0 preconditioned s-step GMRES and ILU0 preconditioned CA-GMRES.
- ▶ The performance of the CA-ILU0(s) reordering based on Metis Nested Dissection and Metis k-way partitioning.

Current and Future Work

- ▶ Implement the CA-ILU0 preconditioner in a parallel environment

Current and Future Work

- ▶ Implement the CA-ILU0 preconditioner in a parallel environment
- ▶ Devise a CA-ILU(k) preconditioner

Current and Future Work

- ▶ Implement the CA-ILU0 preconditioner in a parallel environment
- ▶ Devise a CA-ILU(k) preconditioner
- ▶ Devise other CA-preconditioners

Thank you!