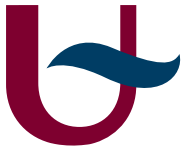


Latency hiding of global reductions in pipelined Krylov methods

On Parallel Performance and Numerical Accuracy of Communication Hiding Pipelined Krylov Subspace Methods for Solving Large Scale Linear Systems

Lawrence Berkeley National Laboratory, CA, US, August 7, 2017

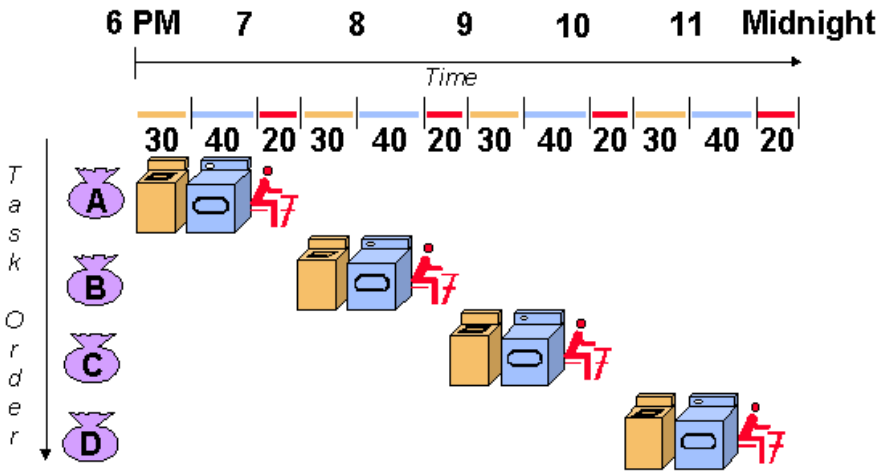
University of Antwerp* [BE] & INRIA Bordeaux† [FR]

Siegfried Cools*, J. Cornelis*, W. Vanroose*, E. F. Yetkin†, E. Agullo†, L. Giraud†

Contact: siegfried.cools@uantwerp.be

Universiteit Antwerpen

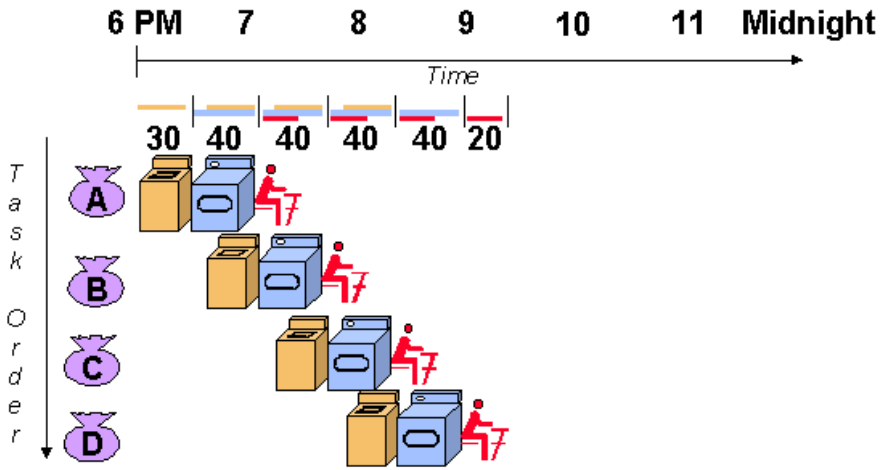
What are we working on?



Classical Krylov subspace method

Washing, *drying* and *ironing*
in classical 'Laundry method'

vs.



Pipelined Krylov subspace method

Latency hiding of global drying
in pipelined 'Laundry method'

Our benefactors

Fellowship 12H4617N
(2016-2019)

Research Foundation
Flanders
Opening new horizons

Observation: increasing gap between computation and communication costs

- Floating point performance steadily increases
- Network latencies only go down marginally
- Memory latencies decline slowly
- Avoid communication:
  trade communication for computations
- Hide communication:
  overlap communication with computations

Work initiated under:
EXascale Algorithms and Advanced Computational Techniques
`http://exa2ct.eu/`

EU FP7 Project - Horizon 2020 - Ran from 2013 to 2016

General idea

Iteratively improve an approximate solution of linear system $Ax = b$,

$$x_i \in x_0 + \mathcal{K}_i(A, r_0) = x_0 + \text{span}\{r_0, Ar_0, A^2 r_0, \ldots, A^{i-1} r_0\}$$

- minimize an error measure over expanding Krylov subspace $\mathcal{K}_i(A, r_0)$
- usually in combination with sparse linear algebra/stencil application
- three building blocks:
  i. dot-product
  ii. SpMV
  iii. axpy

E.g. Conjugate Gradients

Algorithm 1 CG

1: **procedure** CG(A, b, x_0)
2: $r_0 := b - Ax_0$; $p_0 = r_0$
3: **for** $i = 0, \ldots$ **do**
4: $s_i := Ap_i$
5: $\alpha_i := (r_i, r_i) \,/\, (s_i, p_i)$
6: $x_{i+1} := x_i + \alpha_i p_i$
7: $r_{i+1} := r_i - \alpha_i s_i$
8: $\beta_{i+1} := (r_{i+1}, r_{i+1}) \,/\, (r_i, r_i)$
9: $p_{i+1} := r_{i+1} + \beta_{i+1} p_i$
10: **end for**
11: **end procedure**

dot-pr
SpMV
axpy

Classical CG

Algorithm 1 CG

1: **procedure** CG(A, b, x_0)
2: $r_0 := b - Ax_0$; $p_0 = r_0$
3: **for** $i = 0, \ldots$ **do**
4: $s_i := Ap_i$
5: $\alpha_i := (r_i, r_i) / (s_i, p_i)$
6: $x_{i+1} := x_i + \alpha_i p_i$
7: $r_{i+1} := r_i - \alpha_i s_i$
8: $\beta_{i+1} := (r_{i+1}, r_{i+1}) / (r_i, r_i)$
9: $p_{i+1} := r_{i+1} + \beta_{i+1} p_i$
10: **end for**
11: **end procedure**

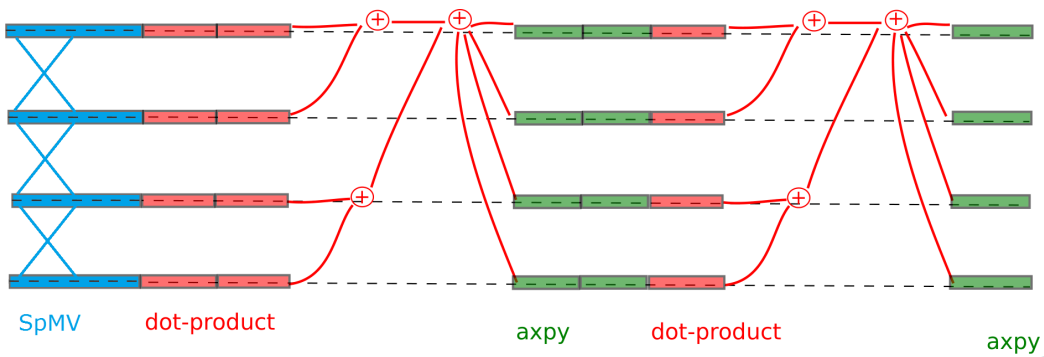
dot-pr
SpMV
axpy

Hestenes & Stiefel (1952)

i. 3 dot-products
- 2 global reduction phases
- latency dominated
- scales as $log_2(\#partitions)$

ii. 1 SpMV
- scales well (minor commun.)
- non-overlapping (sequential to dot-product)

iii. 3 axpy's
- recurrences to avoid SpMV's
- perfectly scalable (no commun.)



Pipelined CG

Algorithm 3 Pipelined CG

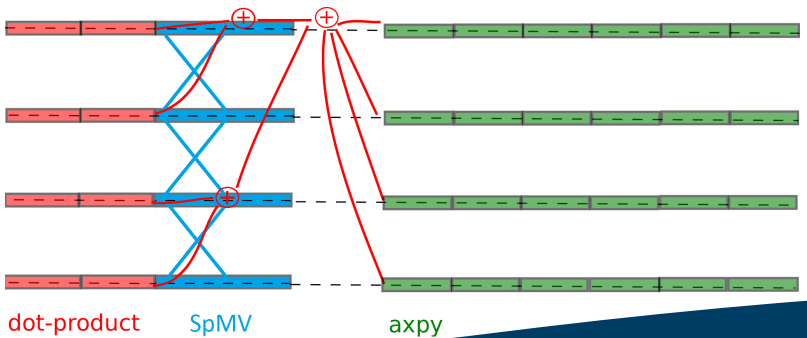
1: **procedure** PIPE-CG(A, b, x_0)
2: $r_0 := b - Ax_0$; $w_0 := Ar_0$
3: **for** $i = 0, \ldots$ **do**
4: $\gamma_i := (r_i, r_i)$
5: $\delta := (w_i, r_i)$
6: $q_i := Aw_i$
7: **if** $i > 0$ **then**
8: $\beta_i := \gamma_i/\gamma_{i-1}$; $\alpha_i := (\delta/\gamma_i - \beta_i/\alpha_{i-1})^{-1}$
9: **else**
10: $\beta_i := 0$; $\alpha := \gamma_i/\delta$
11: **end if**
12: $z_i := q_i + \beta_i z_{i-1}$
13: $s_i := w_i + \beta_i s_{i-1}$
14: $p_i := r_i + \beta_i p_{i-1}$
15: $x_{i+1} := x_i + \alpha_i p_i$
16: $r_{i+1} := r_i - \alpha_i s_i$
17: $w_{i+1} := w_i - \alpha_i z_i$
18: **end for**
19: **end procedure**

dot-pr
SpMV
axpy

Ghysels & Vanroose (2013)

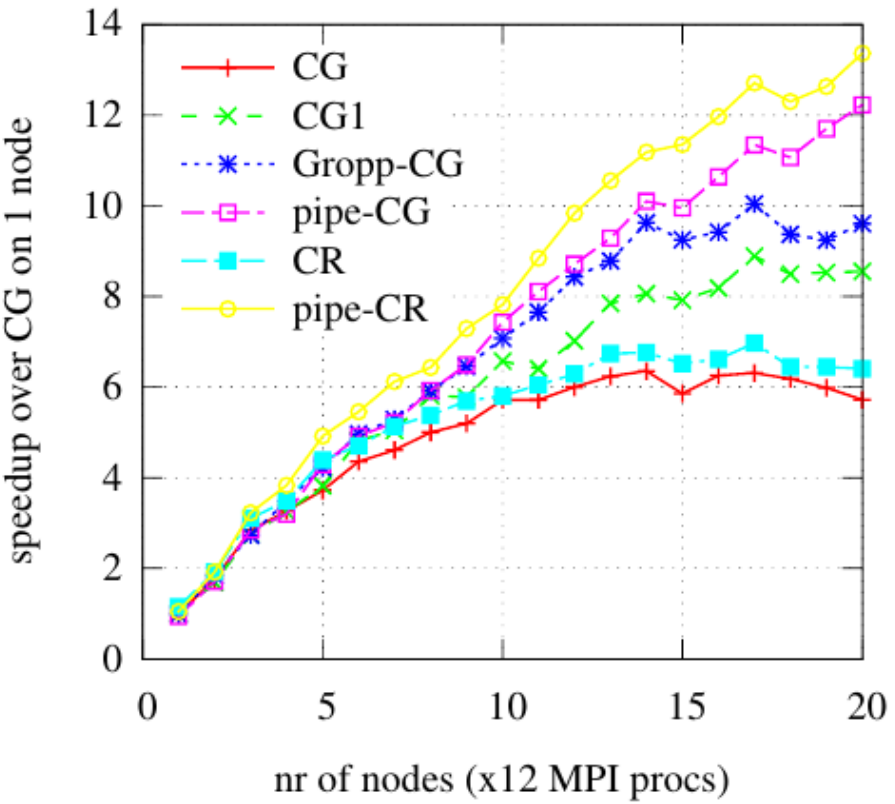
Re-organized version of classical CG for improved parallel performance

- equivalent to CG in exact arithmetic
- Communication avoiding: dot-products are grouped in one global reduction phase (line 4+5)
- Communication hiding: overlap global commun. (line 4+5) with computations (SpMV, line 6)
- 3 extra recurrences for $s_i = Ap_i$, $w_i = Ar_i$, $z_i = As_i$ (line 12+13+17)



Strong scaling experiment

- Hydrostatic ice sheet flow, $100 \times 100 \times 50$ Q1 finite elements, PETSc experiment
- Line search Newton method (rtol=10^{-8}, atol=10^{-15})
- CG preconditioned with block Jacobi with ICC(0) (rtol=10^{-5}, atol=10^{-50})



Ghysels & Vanroose (2013)

Performance breakdown

- max pipe-CG/CG speedup: 2.14×
- max pipe-CG/CG1 speedup: 1.43×
- max pipe-CR/CR speedup: 2.09×

(CG1 = Chronopoulos/Gear CG)
(CR = Conjugate Residuals)

Other pipelined Krylov methods

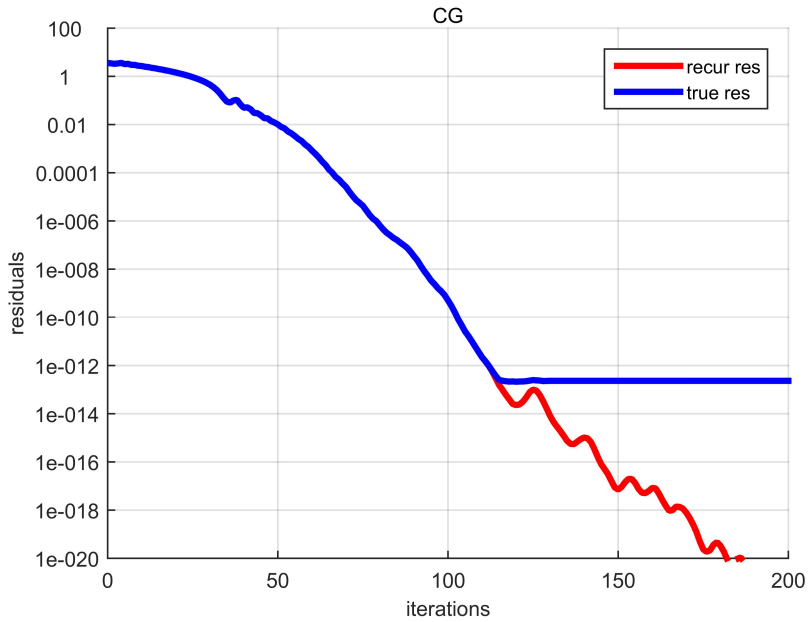
- Pipelined CG — Ghysels, Vanroose (2013)
  - deeper pipelining possible: see further
- Pipelined GMRES — Ghysels et al. (2012)

$$V_{i-\ell+1} = [v_0, v_1, \ldots, v_{i-\ell}]$$
$$Z_{i+1} = [z_0, z_1, \ldots, z_{i-\ell}, \underbrace{z_{i-\ell+1}, \ldots, z_i}_{\ell}]$$

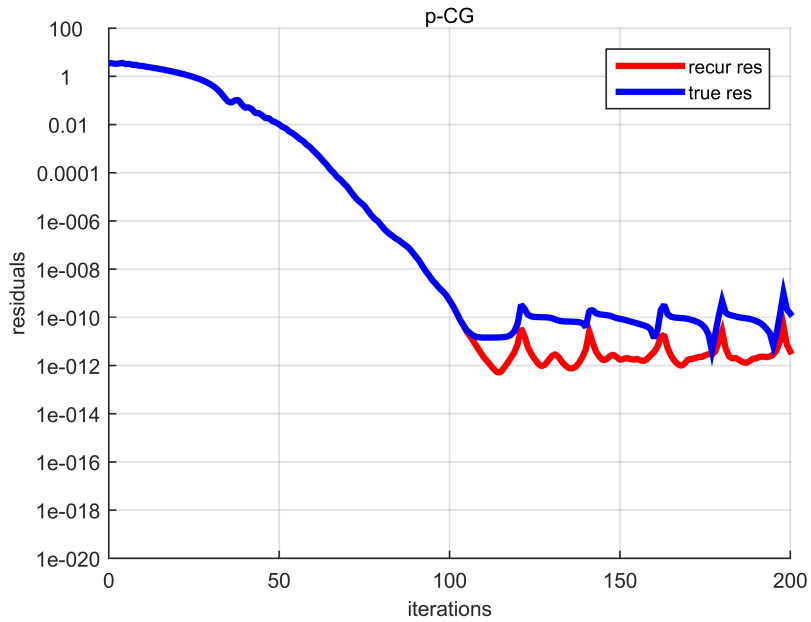
  - compute ℓ new basis vectors for Krylov subspace (SpMVs) during global communication (dot-products).
  - deeper and variable pipelining possible: p(ℓ)-GMRES
- Pipelined BiCGStab — C., Vanroose (2017)
  - non-symmetric operators: see further
- Preconditioned pipelined Krylov methods are available
  - prec-pipe-CG
  - prec-pipe-GMRES
  - prec-pipe-BiCGStab
- Augmented and deflated pipelined Krylov methods are available

Rounding error propagation

Classical CG



Pipelined CG



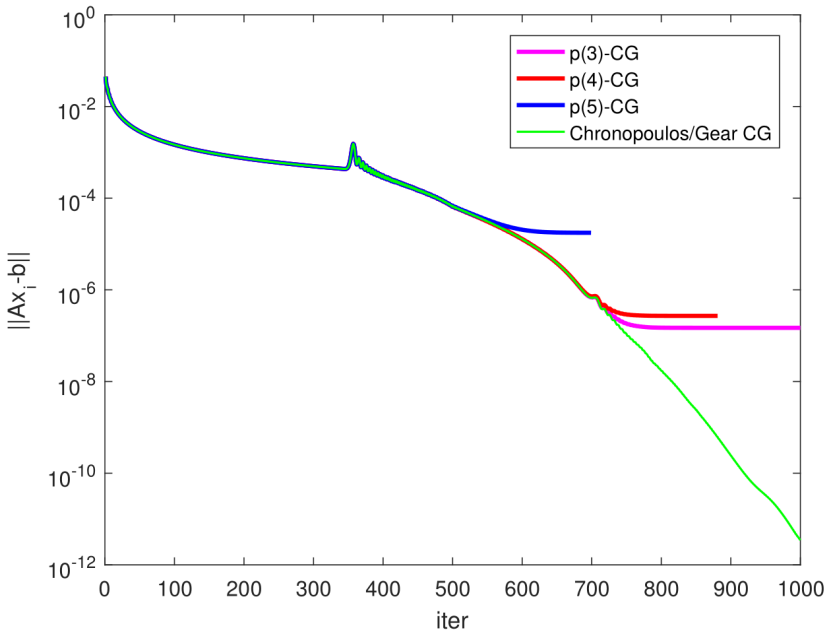
Motivation: pipe-CG loses max. attainable accuracy compared to classical CG

- Model problem: small 2D Laplacian with 2,500 unknowns
- Loss of attainable accuracy is more pronounced for larger systems/longer pipelines

Rounding error propagation

Pipe(ℓ)-CG

J. Cornelis, MaTh (2017)



Motivation: pipe-CG loses max. attainable accuracy compared to classical CG

- Model problem: larger 2D Laplacian with 250,000 unknowns
- Loss of attainable accuracy is more pronounced for larger systems/longer pipelines

Rounding error propagation in CG

Rounding errors due to recursive definition of residual (and auxiliary variables)

$$\begin{aligned} \bar{p}_{i+1} &= \bar{u}_{i+1} + \bar{\beta}_{i+1}\bar{p}_i + \delta_i^p, \\ \bar{x}_{i+1} &= \bar{x}_i + \bar{\alpha}_i\bar{p}_i + \delta_i^x, \\ \bar{r}_{i+1} &= \bar{r}_i - \bar{\alpha}_i A\bar{p}_i + \delta_i^r, \end{aligned}$$

which deviates from the true residual $b - A\bar{x}_i$ in finite precision arithmetics

$$\begin{aligned} f_{i+1} &= (b - A\bar{x}_{i+1}) - \bar{r}_{i+1} \\ &= b - A(\bar{x}_i + \bar{\alpha}_i\bar{p}_i + \delta_i^x) - (\bar{r}_i - \bar{\alpha}_i A\bar{p}_i + \delta_i^r) \\ &= f_i - A\delta_i^x - \delta_i^r. \end{aligned}$$

After i iterations:

$$f_{i+1} = f_0 - \sum_{j=0}^{i} \left(A\delta_j^x + \delta_j^r\right).$$

Only accumulation of local rounding errors in classical CG, no amplification.

Greenbaum (1997), Gutknecht & Strakos (2000)

Rounding errors in pipe-CG

Observation: *rounding error propagation in pipe-CG may be much more dramatic due to additional recurrence relations that all induce rounding errors.*

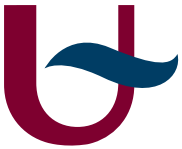
$$\bar{x}_{i+1} = \bar{x}_i + \bar{\alpha}_i \bar{p}_i + \delta_i^x, \qquad \bar{s}_i = \bar{w}_i + \bar{\beta}_i \bar{s}_{i-1} + \delta_i^s,$$
$$\bar{r}_{i+1} = \bar{r}_i - \bar{\alpha}_i \bar{s}_i + \delta_i^r, \qquad \bar{w}_{i+1} = \bar{w}_i - \bar{\alpha}_i \bar{z}_i + \delta_i^w,$$
$$\bar{p}_i = \bar{u}_i + \bar{\beta}_i \bar{p}_{i-1} + \delta_i^p, \qquad \bar{z}_i = A\bar{m}_i + \bar{\beta}_i \bar{z}_{i-1} + \delta_i^z,$$

Residual gap is **coupled** with the gaps on the other auxiliary variables:

$$f_i = (b - A\bar{x}_i) - \bar{r}_i, \quad g_i = A\bar{p}_i - \bar{s}_i, \quad h_i = A\bar{u}_i - \bar{w}_i, \quad j_i = A\bar{q}_i - \bar{z}_i$$

$$\begin{bmatrix} f_{i+1} \\ g_i \\ h_{i+1} \\ j_i \end{bmatrix} = \begin{bmatrix} 1 & -\bar{\alpha}_i\bar{\beta}_i & -\bar{\alpha}_i & 0 \\ 0 & \bar{\beta}_i & 1 & 0 \\ 0 & 0 & 1 & -\bar{\alpha}_i\bar{\beta}_i \\ 0 & 0 & 0 & \bar{\beta}_i \end{bmatrix} \begin{bmatrix} f_i \\ g_{i-1} \\ h_i \\ j_{i-1} \end{bmatrix} + \begin{bmatrix} -A\delta_i^x - \delta_i^r - \bar{\alpha}_i\left(A\delta_i^p - \delta_i^s\right) \\ A\delta_i^p - \delta_i^s \\ A\delta_i^u - \delta_i^w - \bar{\alpha}_i\left(A\delta_i^q - \delta_i^z\right) \\ A\delta_i^q - \delta_i^z \end{bmatrix}.$$

Amplification of local rounding errors possible, depending on α_i's and β_i's.



Rounding error model for CG

Residual gap in iteration i

$\epsilon =$ machine precision

$$f_{i+1} \quad = \quad f_i - A\delta_i^x - \delta_i^r .$$

Error bounds: Local rounding errors $A\delta_i^x + \delta_i^r$ can be bounded by

$$\begin{aligned} \|A\delta_i^x + \delta_i^r\| \quad &\leq \quad \left(\|A\| \, \|\bar{x}_i\| + (\mu\sqrt{n} + 4) \, |\bar{\alpha}_i| \, \|A\| \, \|\bar{p}_i\| + \|\bar{r}_i\|\right) \epsilon \\ &:= \quad e_i^f \, \epsilon. \end{aligned}$$

- often largely overestimates the actual errors

Error estimates: Local rounding errors can be approximated as

$$\|A\delta_i^x + \delta_i^r\| \approx \sqrt{e_i^f} \, \epsilon$$

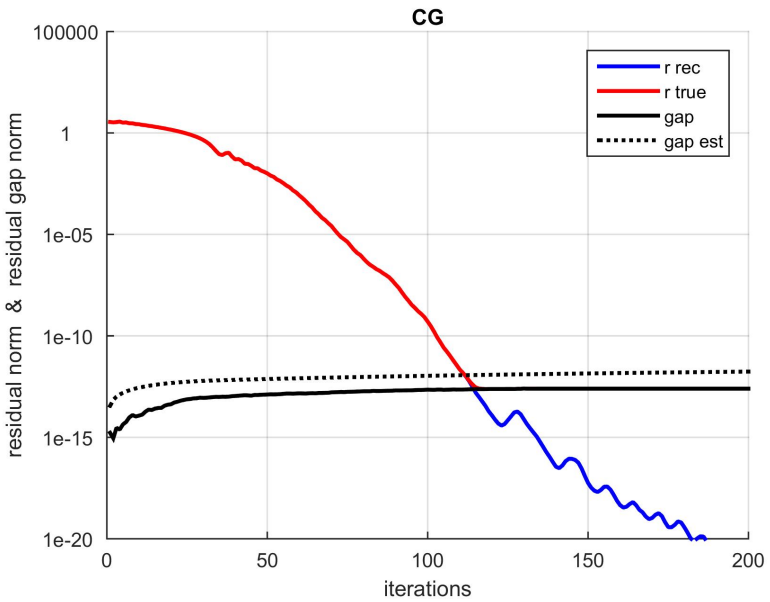
- additional norm computations required
- include in existing global reduction phase to avoid overhead

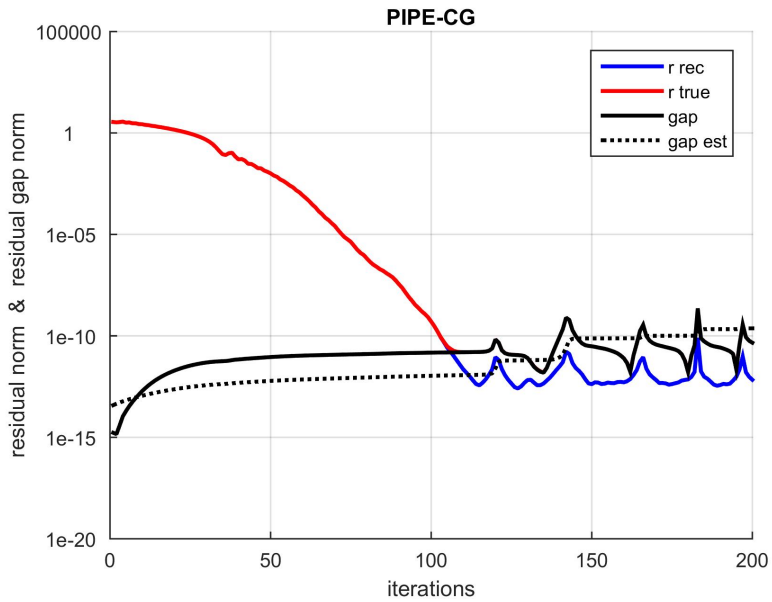
C. & Vanroose (2017)

Rounding errors in pipe-CG

Accumulated rounding error

2D Laplacian, 2,500 unk





- Accumulation of rounding errors causes true residuals to stagnate
- Pipe-CG has reduced maximal attainable accuracy
- Rounding error model: runtime tracking of residual gap using estimate

  Cost? additional DOT-PRs per iteration to compute norms of auxiliary variables

  But: can be included in existing global reduction phase → no additional overhead

Pipe-CG with automated residual replacement

Explicitly replace $\bar{r}_i$, $\bar{s}_i$, $\bar{w}_i$ and $\bar{z}_i$ by their true values in selected iterations:

$$\bar{r}_{i+1} = \mathrm{fl}(b - A\bar{x}_{i+1}), \qquad \bar{w}_{i+1} = \mathrm{fl}(A\bar{u}_{i+1}),$$
$$\bar{s}_i = \mathrm{fl}(A\bar{p}_i), \qquad \bar{z}_i = \mathrm{fl}(A\bar{q}_i).$$

Residual replacement criterion:

$$\|f_{i-1}\| \leq \tau \|\bar{r}_{i-1}\| \quad \text{and} \quad \|f_i\| > \tau \|\bar{r}_i\|.$$

with $\tau = \sqrt{\epsilon}$.

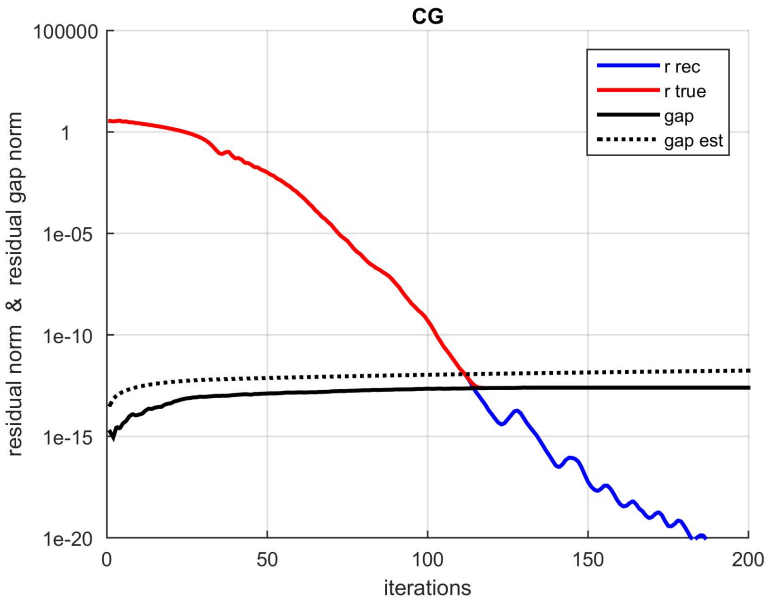
- Sleijpen & Van der Vorst 1996
- Tong & Ye, 1999
- Van der Vorst & Ye, 2000

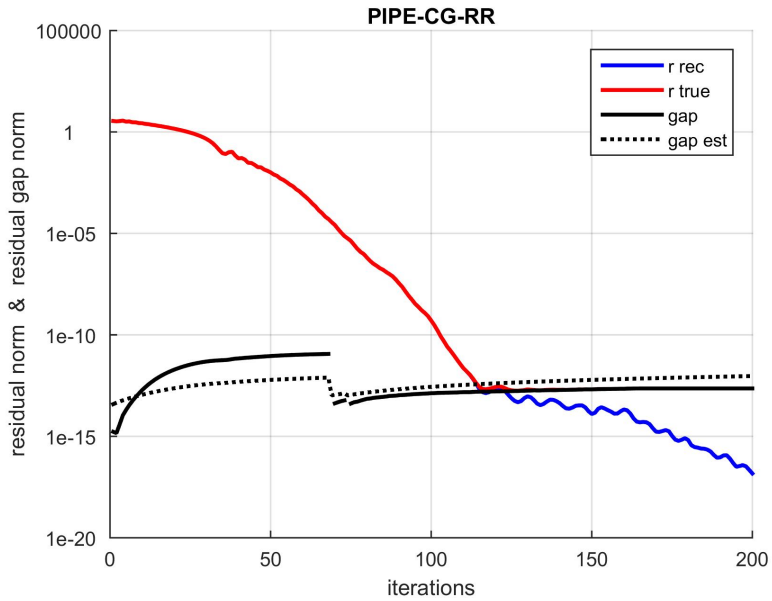
Estimate for gap $\|f_i\|$ can be computed at runtime (without additional overhead), so fully automated replacement strategy is possible for pipe-CG.

Pipe-CG with automated residual replacement

Accumulated rounding error

2D Laplacian, 2,500 unk





- Pipe-CG-rr = pipe-CG with residual replacement based on rounding error model

  Cost? 4 additional SpMV's per replacement step

- Replacement criterion ensures:
  (1) number of replacements is limited,
  (2) only replace when $\|r_i\|$ is sufficiently large (Krylov convergence is not affected)
  Tong & Ye, 1999

Numerical results: attainable accuracy

Laplacian 2D: convergence tests on Laplacian problems

$\bar{x}_0 = 0$		CG		CG-CG		p-CG		p-CG-rr		
Matrix	n	iter	relres relerr	iter	relres relerr	iter	relres relerr	iter	relres relerr	rr
lapl50	2,500	128	7.8e-15 6.4e-15	127	8.1e-15 5.7e-15	118	1.5e-12 1.1e-12	125	9.1e-15 2.9e-14	3
lapl100	10,000	254	1.6e-14 1.4e-14	256	1.6e-14 1.4e-14	228	9.1e-12 6.5e-12	272	1.2e-14 1.8e-14	6
lapl200	40,000	490	3.1e-14 3.7e-14	487	3.2e-14 3.6e-14	439	5.4e-11 5.3e-11	536	2.5e-14 3.7e-14	11
lapl400	160,000	959	6.2e-14 1.0e-13	958	6.4e-14 5.6e-14	807	3.0e-10 3.4e-10	957	4.6e-14 1.8e-13	23
lapl800	640,000	1883	1.2e-13 2.7e-13	1877	1.3e-13 8.2e-13	1524	1.4e-10 2.0e-09	1876	1.1e-13 2.1e-13	53
$\bar{x}_0 = \text{rand}(n, 1)$		CG		CG-CG		p-CG		p-CG-rr		
Matrix	n	iter	relres relerr	iter	relres relerr	iter	relres relerr	iter	relres relerr	rr
lapl50	2,500	232	9.0e-14 4.9e-14	237	1.1e-13 6.4e-14	197	1.3e-10 1.1e-10	227	1.9e-14 7.4e-14	6
lapl100	10,000	444	2.9e-13 1.6e-13	449	3.8e-13 2.1e-13	367	1.5e-09 1.3e-09	483	1.6e-14 1.5e-14	10
lapl200	40,000	881	1.3e-12 6.2e-13	883	1.6e-12 7.8e-13	685	1.7e-08 1.6e-08	952	3.3e-14 3.9e-14	20
lapl400	160,000	1676	4.9e-12 2.3e-12	1714	6.1e-12 2.9e-12	1220	1.8e-07 1.8e-07	1846	1.1e-13 1.5e-13	35
lapl800	640,000	3339	2.1e-11 9.6e-12	3249	2.5e-11 1.2e-11	2225	1.9e-06 2.1e-06	3435	2.1e-12 2.4e-12	65

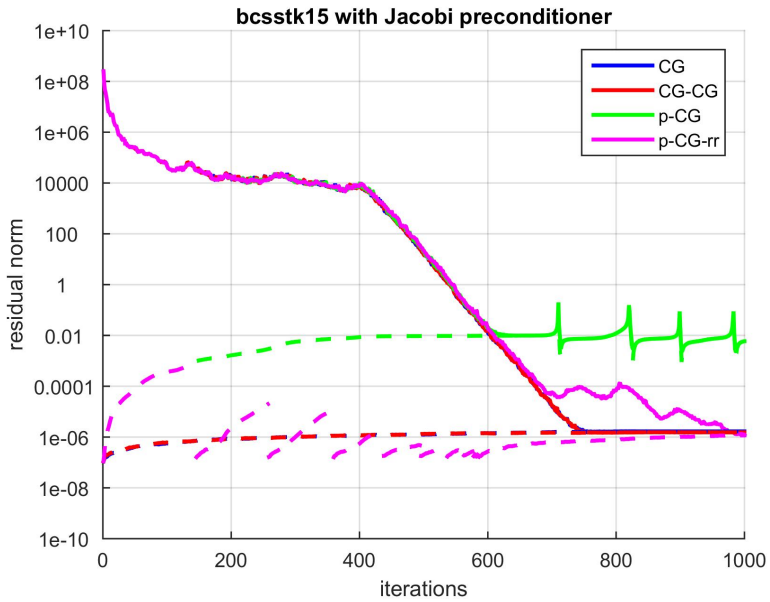
Numerical results: attainable accuracy

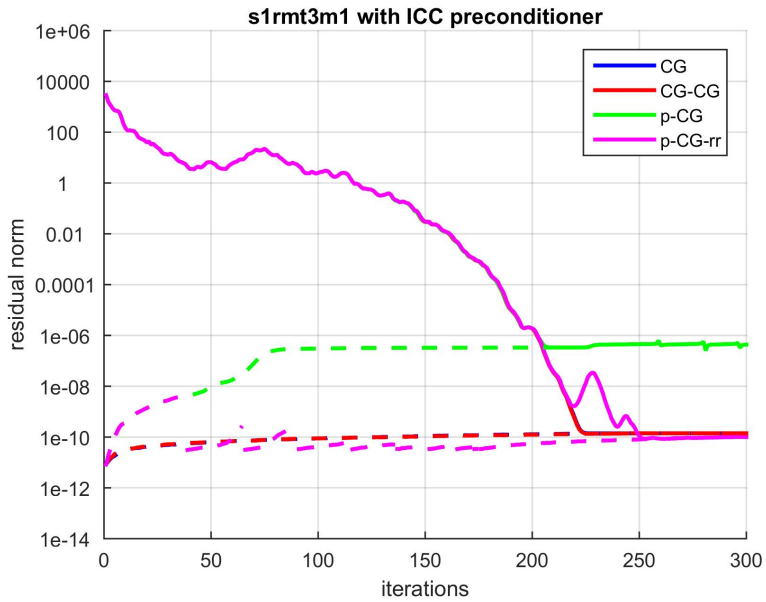
MatrixMarket collection: convergence tests on all non-diagonal SPD matrices

Matrix	Prec	$\kappa(A)$	n	$\#nnz$	CG		CG-CG		p-CG		p-CG-rr		
					iter	relres	iter	relres	iter	relres	iter	relres	rr
bcsstk14	JAC	1.3e+10	1806	63,454	650	7.6e-16	658	7.1e-16	506	5.2e-12	658	5.2e-16	9
bcsstk15	JAC	8.0e+09	3948	117,816	772	3.7e-15	785	3.5e-15	646	2.3e-11	974	4.0e-15	10
bcsstk16	JAC	65	4884	290,378	298	3.5e-15	300	4.0e-15	261	8.7e-12	301	2.1e-15	4
bcsstk17	JAC	65	10,974	428,650	3547	1.0e-14	3428	1.7e-14	2913	2.8e-09	4508	1.2e-14	54
bcsstk18	JAC	65	11,948	149,090	2299	2.2e-15	2294	2.1e-15	1590	2.9e-11	2400	1.5e-15	50
bcsstk27	JAC	7.7e+04	1224	56,126	345	3.2e-15	345	4.0e-15	295	8.0e-12	342	2.7e-15	6
gr_30_30	-	3.8e+02	900	7744	56	2.7e-15	55	3.1e-15	52	2.0e-13	61	3.0e-15	2
nos1	*ICC	2.5e+07	237	1017	301	1.3e-14	338	1.2e-14	337	2.6e-10	968	1.9e-14	21
nos2	*ICC	6.3e+09	957	4137	3180	8.3e-14	3292	1.1e-13	2656	1.2e-07	4429	2.7e-11	113
nos3	ICC	7.3e+04	960	15,844	64	1.0e-14	63	1.1e-14	59	1.0e-12	61	2.5e-14	3
nos4	ICC	2.7e+03	100	594	31	1.9e-15	31	1.9e-15	29	4.0e-14	33	1.3e-15	2
nos5	ICC	2.9e+04	468	5172	63	3.2e-16	64	3.4e-16	58	4.3e-14	65	2.3e-16	2
nos6	ICC	8.0e+06	675	3255	34	5.1e-15	35	6.2e-15	31	5.5e-11	33	1.0e-14	2
nos7	ICC	4.1e+09	729	4617	29	4.0e-14	31	2.8e-14	29	4.5e-14	29	3.0e-14	3
s1rmq4m1	ICC	1.8e+06	5489	262,411	122	4.3e-15	122	4.6e-15	114	5.5e-12	135	3.7e-15	6
s1rmt3m1	ICC	2.5e+06	5489	217,651	229	9.3e-15	228	8.7e-15	213	2.2e-11	240	1.7e-14	9
s2rmq4m1	*ICC	1.8e+08	5489	263,351	370	6.7e-15	387	7.3e-15	333	2.7e-10	349	2.5e-13	25
s2rmt3m1	ICC	2.5e+08	5489	217,681	285	8.7e-15	283	1.0e-14	250	7.3e-10	425	8.7e-15	17
s3dkq4m2	*ICC	1.9e+11	90,449	2,455,670	-	1.9e-08	-	2.1e-08	-	2.8e-07	-	5.6e-08	199
s3dkt3m2	*ICC	3.6e+11	90,449	1,921,955	-	2.9e-07	-	2.9e-07	-	3.5e-07	-	2.9e-07	252
s3rmq4m1	*ICC	1.8e+10	5489	262,943	1651	1.5e-14	1789	1.6e-14	1716	2.6e-08	1602	5.3e-10	154
s3rmt3m1	*ICC	2.5e+10	5489	217,669	2282	2.7e-14	2559	2.9e-14	2709	9.3e-08	3448	8.0e-10	149
s3rmt3m3	*ICC	2.4e+10	5357	207,123	2862	3.3e-14	2798	3.4e-14	3378	2.0e-07	2556	7.1e-11	248

Numerical results: attainable accuracy

MatrixMarket collection: convergence tests on selected SPD matrices





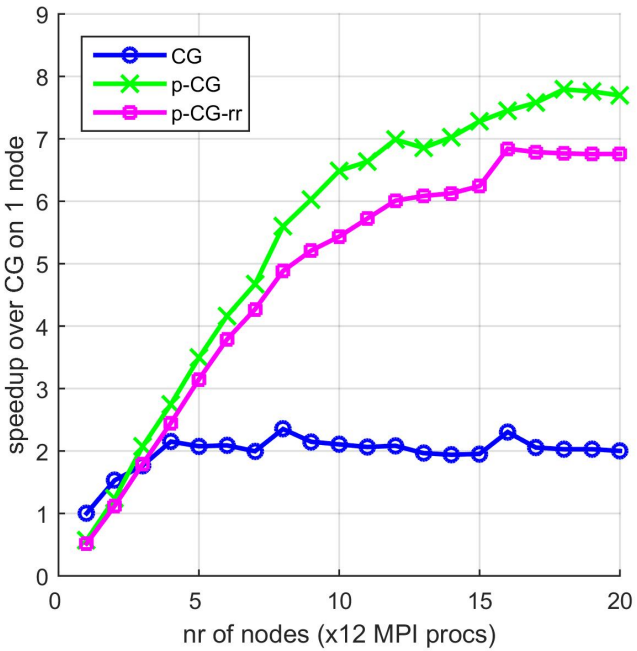
Note: delay of convergence due to loss of Krylov basis orthogonality may occur.

Strakos & Tichy, 2002

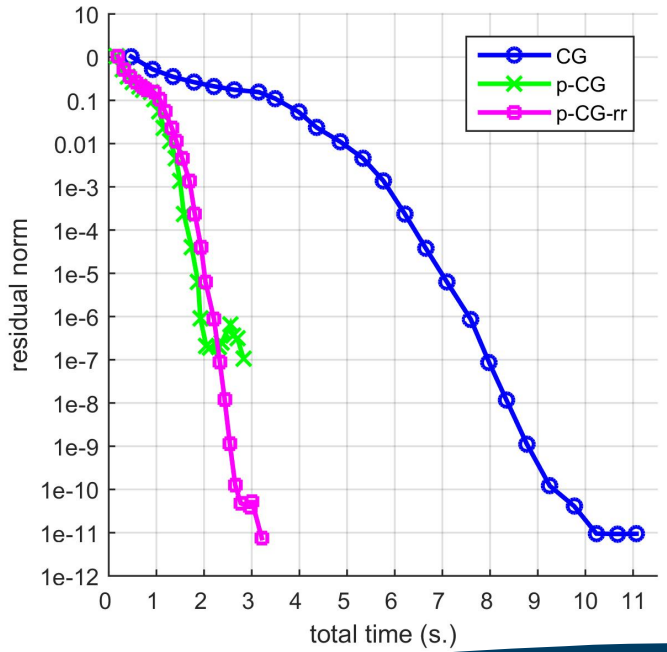
Numerical results: strong scaling

- PETSc implementation using MPICH-3.1.3 communication
- Benchmark problem: 2D Laplacian model, 1,000,000 unknowns
- System specs: 20 nodes, two 6-core Intel Xeon X5660 Nehalem 2.8GHz CPUs/node

Speedup over single-node CG (12-240 cores)



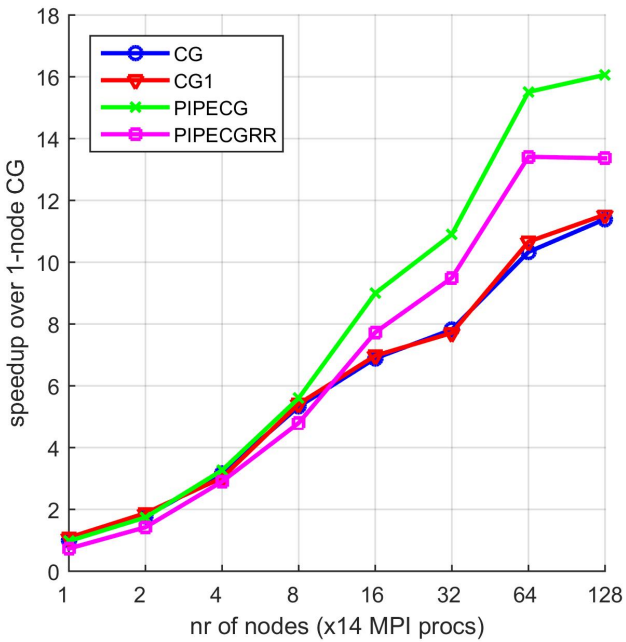
Accuracy i.f.o. total time spent (240 cores)



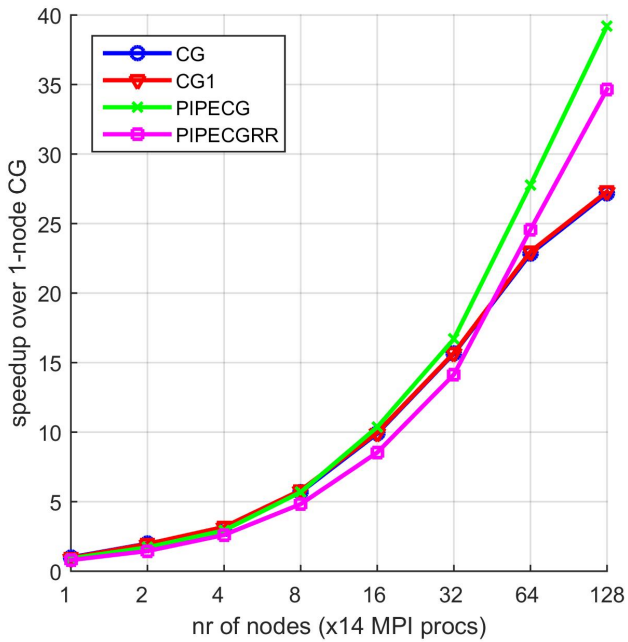
Numerical results: strong scaling

- PETSc implementation using MPICH-3.3a2 communication
- Benchmark problem: 3D ice sheet flow, $150 \times 150 \times 100$ / $500 \times 500 \times 50$ Q1 FE
- System specs: 128 nodes, two 14-core Intel Xeon E5-2680v4 2.4GHz CPUs/node

Speedup over single-node CG (2,250,000 unk)



Speedup over single-node CG (12,500,000 unk)



Shifted pipelined CG

Algorithm 3 Shifted pipelined CG

1: **procedure** P-CG-SH($A, M^{-1}, b, x_0, \sigma$)
2: $r_0 := b - Ax_0$; $u_0 := M^{-1} r_0$
3: $w_0 := Au_0 - \sigma r_0$
4: **for** $i = 0, \ldots$ **do**
5: $\gamma_i := (r_i, u_i)$; $\delta := (w_i + \sigma r_i, u_i)$
6: $m_i := M^{-1} w_i$
7: $n_i := Am_i$
8: **if** $i > 0$ **then**
9: $\beta_i := \gamma_i / \gamma_{i-1}$
10: $\alpha_i := (\delta/\gamma_i - \beta_i/\alpha_{i-1})^{-1}$
11: **else**
12: $\beta_i := 0$; $\alpha_i := \gamma_i / \delta$
13: **end if**
14: $z_i := n_i + \beta_i z_{i-1}$
15: $q_i := m_i + \beta_i q_{i-1}$
16: $s_i := w_i + \beta_i s_{i-1}$
17: $t_i := r_i + \beta_i t_{i-1}$
18: $p_i := u_i + \beta_i p_{i-1}$
19: $x_{i+1} := x_i + \alpha_i p_i$
20: $r_{i+1} := r_i - \alpha_i s_i - \alpha_i \sigma t_i$
21: $u_{i+1} := u_i - \alpha_i q_i - \alpha_i \sigma p_i$
22: $w_{i+1} := w_i - \alpha_i z_i$
23: **end for**

C. & Vanroose (2017)

Define selected auxiliary variables using a shifted matrix:

$$w_i := \left(AM^{-1} - \sigma I\right) r_i = Au_i - \sigma r_i,$$
$$s_i := \left(AM^{-1} - \sigma I\right) t_i = Ap_i - \sigma t_i,$$

and reformulate recursions accordingly.

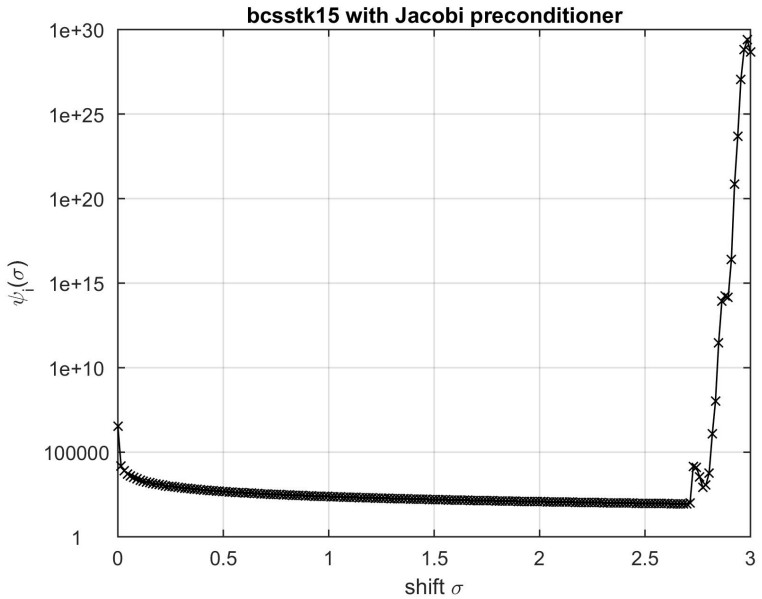
Rounding error propagation:

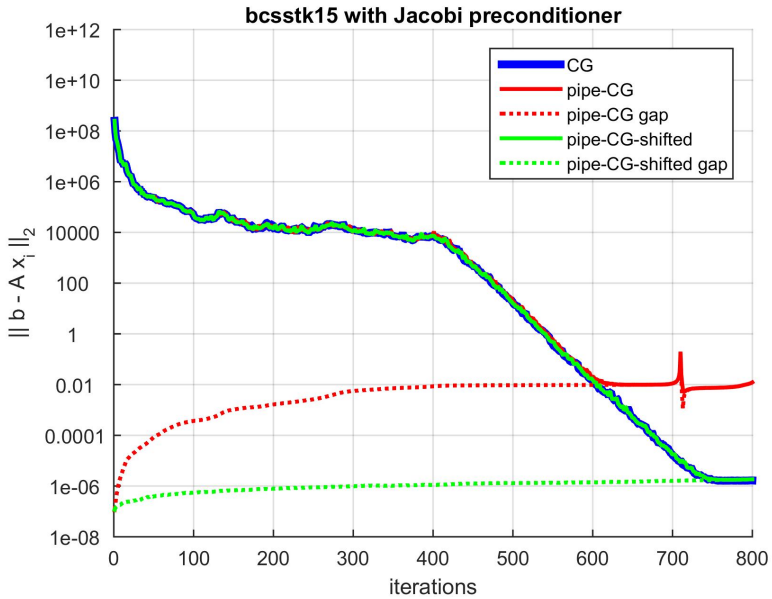
$$\begin{bmatrix} f_{i+1} \\ g_i \\ h_{i+1} \\ j_i \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & -\bar{\alpha}_i\bar{\beta}_i & -\bar{\alpha}_i & 0 \\ 0 & \bar{\beta}_i & 1 & 0 \\ 0 & -\bar{\alpha}_i\bar{\beta}_i\sigma & 1-\bar{\alpha}_i\sigma & -\bar{\alpha}_i\bar{\beta}_i \\ 0 & 0 & 0 & \bar{\beta}_i \end{bmatrix}}_{:= P_i(\sigma)} \begin{bmatrix} f_i \\ g_{i-1} \\ h_i \\ j_{i-1} \end{bmatrix} + \begin{bmatrix} \epsilon_i^f \\ \epsilon_i^g \\ \epsilon_i^h \\ \epsilon_i^j \end{bmatrix}$$

Key idea: *Proper choice of shift counteracts rounding error amplification and stabilizes pipe-CG.*

Shifted pipelined CG

MatrixMarket collection: convergence tests on selected SPD matrices





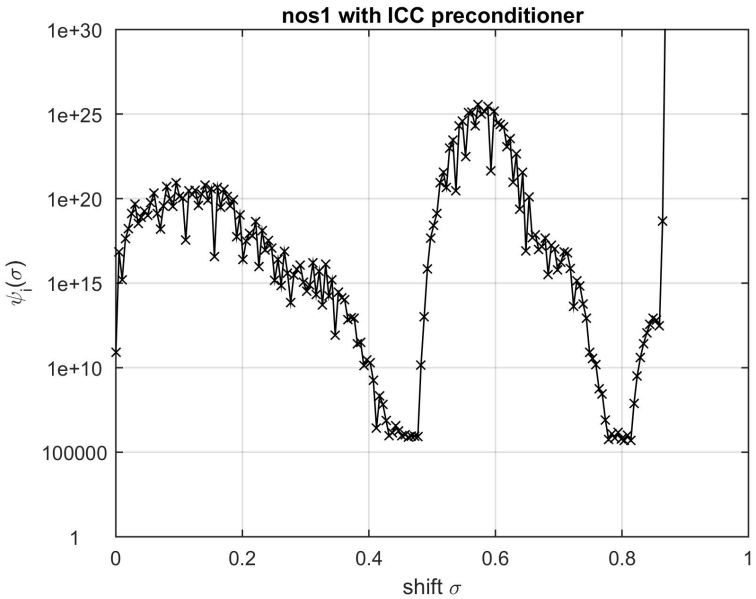
$$\psi_i(\sigma) = \max_{1 \le j \le i} \left\| \prod_{k=j}^{i} P_k(\sigma) \right\|_2$$

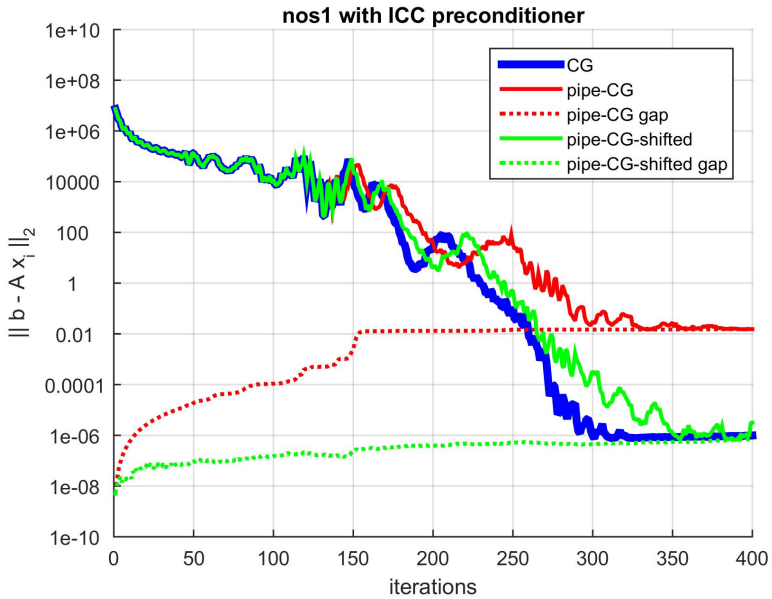
Remarks:

- choice of shift is based on an a posteriori estimate (α_i's and β_i's required)
- delay of convergence possible (cf. pipe-CG)

Shifted pipelined CG

MatrixMarket collection: convergence tests on selected SPD matrices





$$\psi_i(\sigma) = \max_{1 \le j \le i} \left\| \prod_{k=j}^{i} P_k(\sigma) \right\|_2$$

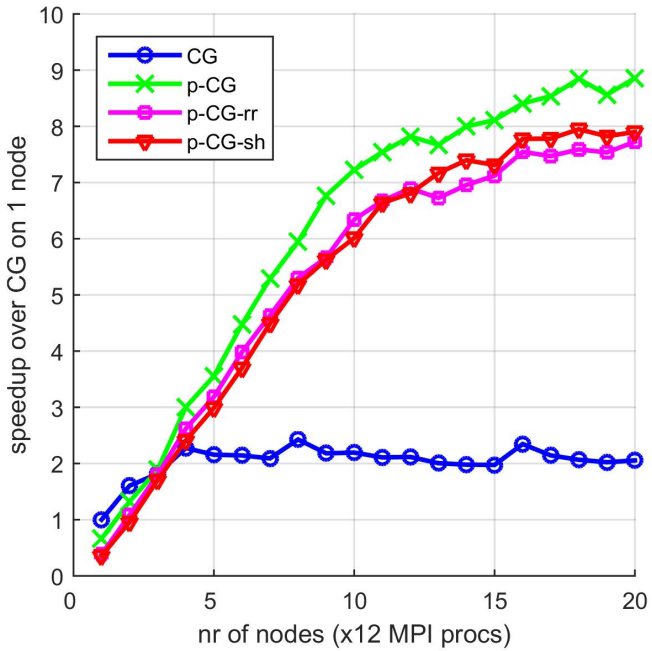
Remarks:

- choice of shift is based on an a posteriori estimate (α_i's and β_i's required)
- delay of convergence possible (cf. pipe-CG)

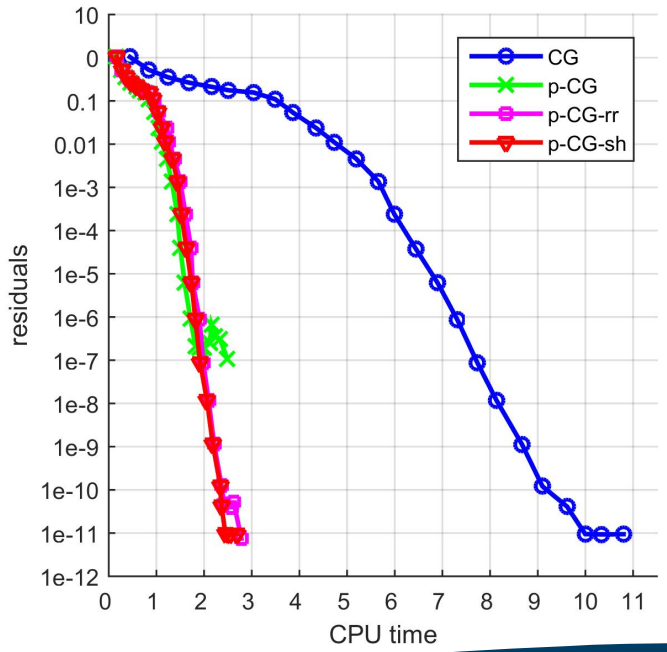
Numerical results: strong scaling

- PETSc implementation using MPICH-3.1.3 communication
- Benchmark problem: 2D Laplacian model, 1,000,000 unknowns
- System specs: 20 nodes, two 6-core Intel Xeon X5660 Nehalem 2.8GHz CPUs/node

Speedup over single-node CG (12-240 cores)

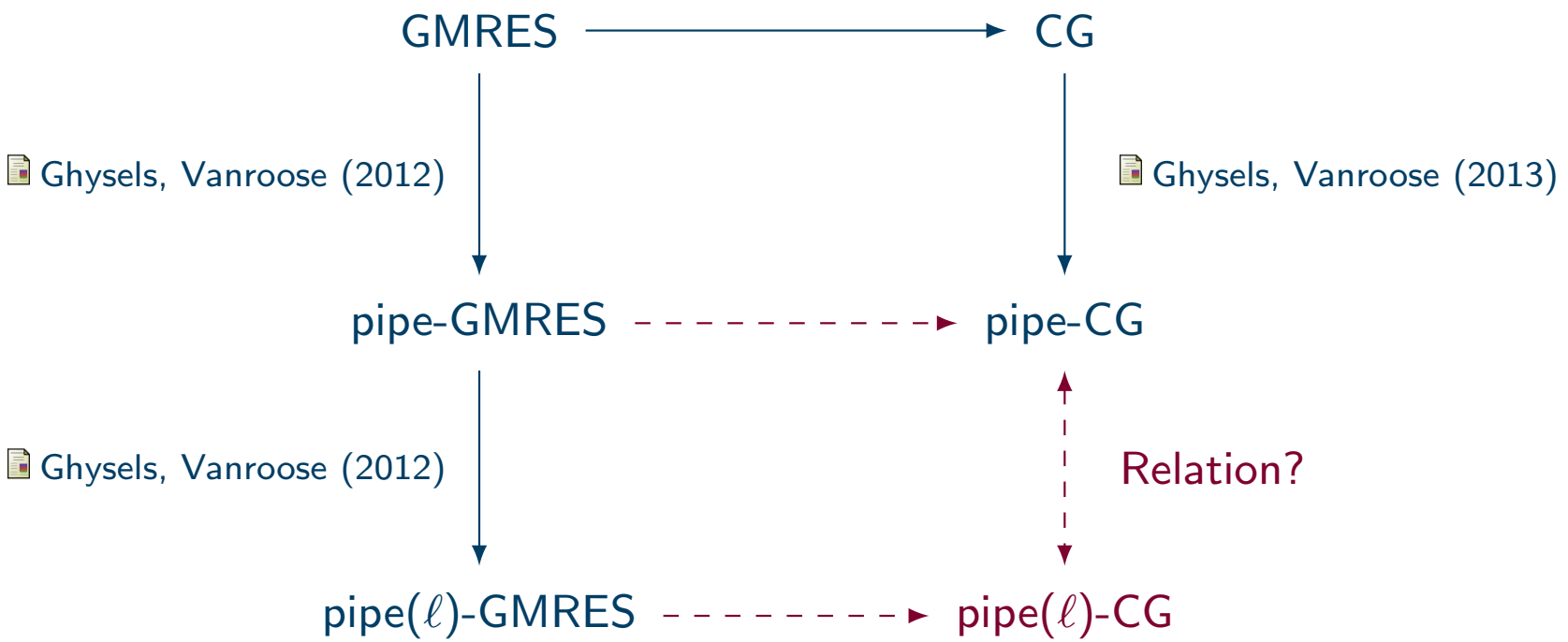


Accuracy i.f.o. total time spent (240 cores)



Commutative diagram?

Saad (2003)



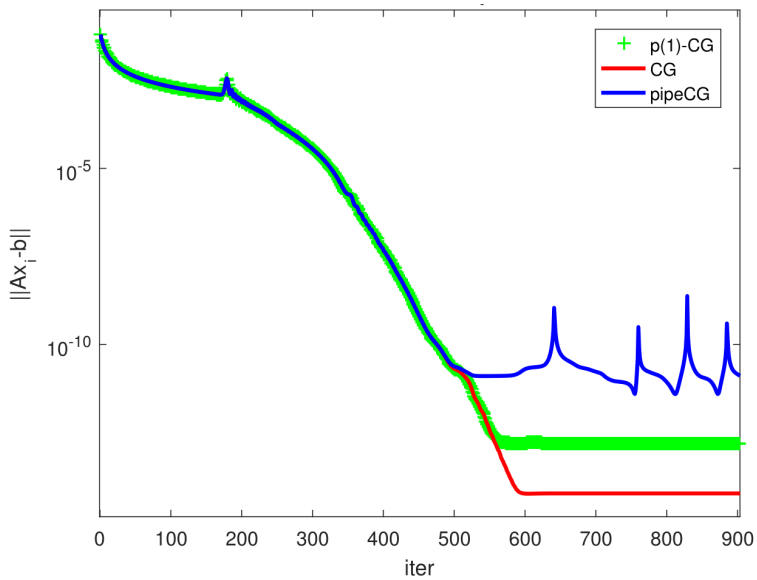
Cornelis, MaTh (2017)

Convergence comparison:

pipe-CG vs. pipe(1)-CG

2D Laplacian, 62,500 unk

J. Cornelis, MaTh (2017)



All CG variants are equivalent in exact arithmetic; however, pipe-CG and pipe(1)-CG are not equivalent in finite precision.

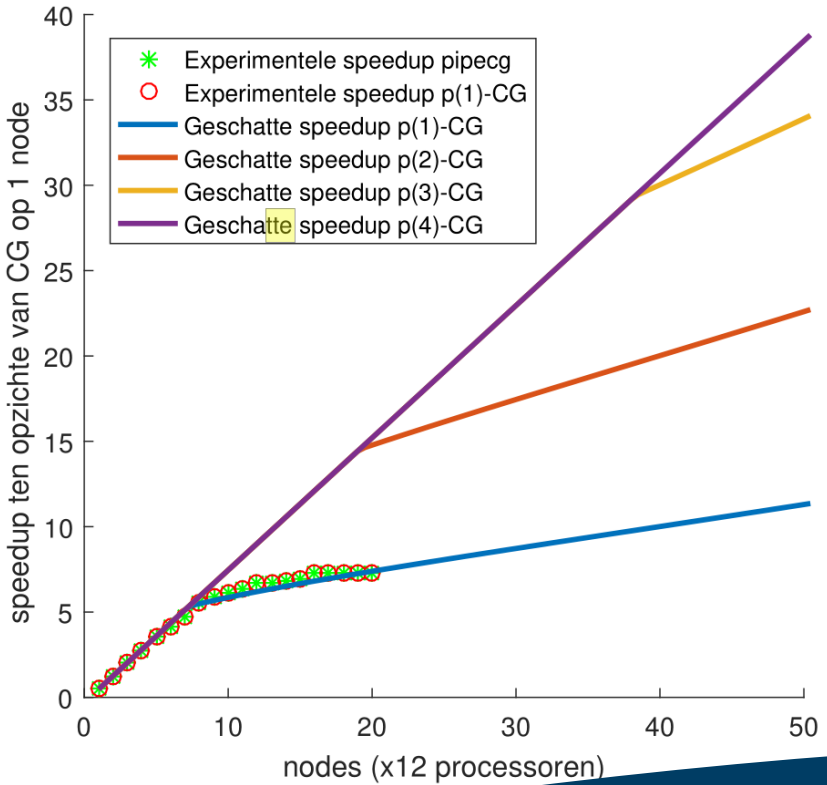
- pipe(1)-CG appears to be more robust w.r.t. local rounding errors
- pipe(1)-CG has less AXPYs compared to pipe-CG

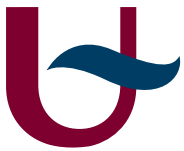
Numerical results: strong scaling

- PETSc implementation using MPICH-3.1.3 communication
- Benchmark problem: 2D Laplacian model, 1,000,000 unknowns
- System specs: 20 nodes, two 6-core Intel Xeon X5660 Nehalem 2.8GHz CPUs/node

Measured/predicted speedup over 1-node CG
240 cores/600 cores

J. Cornelis, MaTh (2017)





BiCGStab

Algorithm 4 Standard BiCGStab

```
1:  function BICGSTAB(A, b, x_0)
2:      r_0 := b − Ax_0; p_0 := r_0
3:      for i = 0, ... do
4:          s_i := Ap_i
5:          compute (r_0, s_i)
6:          α_i := (r_0, r_i) / (r_0, s_i)
7:          q_i := r_i − α_i s_i
8:          y_i := Aq_i
9:          compute (q_i, y_i) ; (y_i, y_i)
10:         ω_i := (q_i, y_i) / (y_i, y_i)
11:         x_{i+1} := x_i + α_i p_i + ω_i q_i
12:         r_{i+1} := q_i − ω_i y_i
13:         compute (r_0, r_{i+1})
14:         β_i := (α_i/ω_i) (r_0, r_{i+1}) / (r_0, r_i)
15:         p_{i+1} := r_{i+1} + β_i (p_i − ω_i s_i)
16:     end for
17: end function
```

dot-prod
SpMV
axpy

Traditional BiCGStab:

(non-preconditioned)

Global communication

- 3 global reduction phases

Local communication

- 2 non-overlapping SpMVs

No communication

- 4 recurrences

General two-step framework for deriving pipelined Krylov methods:

Step 1. Avoiding communication: merge global reductions

Step 2. Hiding communication: overlap SpMVs & global reductions

Step 1. Avoiding global communication

Algorithm 4 Standard BiCGStab

1: **function** BICGSTAB(A, b, x_0)
2: $r_0 := b - Ax_0$; $p_0 := r_0$
3: **for** $i = 0, \ldots$ **do**
4: $s_i := Ap_i$
5: **compute** (r_0, s_i)
6: $\alpha_i := (r_0, r_i) \,/\, (r_0, s_i)$
7: $q_i := r_i - \alpha_i s_i$
8: $y_i := Aq_i$
9: **compute** (q_i, y_i) ; (y_i, y_i)
10: $\omega_i := (q_i, y_i) \,/\, (y_i, y_i)$
11: $x_{i+1} := x_i + \alpha_i p_i + \omega_i q_i$
12: $r_{i+1} := q_i - \omega_i y_i$
13: **compute** (r_0, r_{i+1})
14: $\beta_i := (\alpha_i/\omega_i)\,(r_0, r_{i+1}) \,/\, (r_0, r_i)$
15: $p_{i+1} := r_{i+1} + \beta_i\,(p_i - \omega_i s_i)$
16: **end for**
17: **end function**

dot-prod
SpMV
axpy

(a) *Identify* two global comm. phases for merger (lines 5-6 & 13-14)

(b) *Rewrite* SpMV as recurrence:

$$s_i = Ap_i = w_i + \beta_{i-1}\,(s_{i-1} - \omega_{i-1} z_{i-1})\,,$$

define $w_i := Ar_i$, $z_i := As_i$ and note that $y_i := w_i - \alpha_i z_i$

(c) *Rewrite* dot-product using (b):

$$(r_0, s_i) = (r_0, w_i + \beta_{i-1}\,(s_{i-1} - \omega_{i-1} z_{i-1}))\,,$$

independent of interlying variables

(d) *Move* dot-product (lines 5-6) upward and merge with existing global comm. phase (lines 13-14)

CA-BiCGStab

Algorithm 5 Communication avoiding BiCGStab

1: **function** CA-BICGSTAB(A, b, x_0)
2: $r_0 := b - Ax_0$; $w_0 := Ar_0$; $\alpha_0 := (r_0, r_0) \,/\, (r_0, w_0)$; $\beta_{-1} := 0$
3: **for** $i = 0, \ldots$ **do**
4: $p_i := r_i + \beta_{i-1} (p_{i-1} - \omega_{i-1} s_{i-1})$
5: $s_i := w_i + \beta_{i-1} (s_{i-1} - \omega_{i-1} z_{i-1})$
6: $z_i := As_i$
7: $q_i := r_i - \alpha_i s_i$ **dot-prod**
8: $y_i := w_i - \alpha_i z_i$ **SpMV**
9: **compute** (q_i, y_i) ; (y_i, y_i) **axpy**
10: $\omega_i := (q_i, y_i) \,/\, (y_i, y_i)$
11: $x_{i+1} := x_i + \alpha_i p_i + \omega_i q_i$
12: $r_{i+1} := q_i - \omega_i y_i$
13: $w_{i+1} := Ar_{i+1}$
14: **compute** (r_0, r_{i+1}) ; (r_0, w_{i+1}) ; (r_0, s_i) ; (r_0, z_i)
15: $\beta_i := (\alpha_i / \omega_i) \, (r_0, r_{i+1}) \,/\, (r_0, r_i)$
16: $\alpha_{i+1} := (r_0, r_{i+1}) \,/\, ((r_0, w_{i+1}) + \beta_i (r_0, s_i) - \beta_i \omega_i (r_0, z_i))$
17: **end for**
18: **end function**

Communication-avoiding BiCGStab:
(non-preconditioned)

Global communication

- 2 global red. phases (vs. 3)

Local communication

- 2 non-overlapping SpMVs

No communication

- 6 recurrences (vs. 4)

Status after Step 1:

no. global comm. phases reduced from 3 to 2, at the cost of 2 additional axpys

Note: further reduction from 2 to 1 global comm. phase possible, but not recommended (see later).

Step 2. Hiding global communication

Algorithm 5 Communication avoiding BiCGStab

1: **function** CA-BICGSTAB(A, b, x_0)
2: $r_0 := b - Ax_0$; $w_0 := Ar_0$; $\alpha_0 := (r_0, r_0) \,/\, (r_0, w_0)$; $\beta_{-1} := 0$
3: **for** $i = 0, \ldots$ **do**
4: $p_i := r_i + \beta_{i-1} (p_{i-1} - \omega_{i-1} s_{i-1})$
5: $s_i := w_i + \beta_{i-1} (s_{i-1} - \omega_{i-1} z_{i-1})$
6: $z_i := As_i$
7: $q_i := r_i - \alpha_i s_i$ **dot-prod**
8: $y_i := w_i - \alpha_i z_i$ **SpMV**
9: **compute** (q_i, y_i) ; (y_i, y_i) **axpy**
10: $\omega_i := (q_i, y_i) \,/\, (y_i, y_i)$
11: $x_{i+1} := x_i + \alpha_i p_i + \omega_i q_i$
12: $r_{i+1} := q_i - \omega_i y_i$
13: $w_{i+1} := Ar_{i+1}$
14: **compute** (r_0, r_{i+1}) ; (r_0, w_{i+1}) ; (r_0, s_i) ; (r_0, z_i)
15: $\beta_i := (\alpha_i / \omega_i) \, (r_0, r_{i+1}) \,/\, (r_0, r_i)$
16: $\alpha_{i+1} := (r_0, r_{i+1}) \,/\, ((r_0, w_{i+1}) + \beta_i (r_0, s_i) - \beta_i \omega_i (r_0, z_i))$
17: **end for**
18: **end function**

(a) *Identify* SpMV / global reduction pairs (lines 6 & 9 and 13 & 14)

(b) *Rewrite* SpMVs as recurrences:

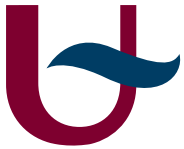
$$z_i := As_i = t_i + \beta_{i-1} (z_{i-1} - \omega_{i-1} v_{i-1}),$$
$$w_{i+1} := Ar_{i+1} = y_i - \omega_i (t_i - \alpha_i v_i),$$

define $t_i := Aw_i$, $v_i := Az_i$

(c) *Check* SpMV / global reduction pairwise dependencies:

- line 9 independent of v_i? **yes**
- line 14 indep. of t_{i+1}? **yes**

(d) *Insert* new SpMVs *below* corresponding global comm. phases



pipe-BiCGStab

Algorithm 6 Pipelined BiCGStab

1: **function** PIPE-BICGSTAB(A, b, x_0)
2: $r_0 := b - Ax_0$; $w_0 := Ar_0$; $t_0 := Aw_0$;
3: **for** $i = 0, \ldots$ **do**
4: $p_i := r_i + \beta_{i-1}(p_{i-1} - \omega_{i-1}s_{i-1})$
5: $s_i := w_i + \beta_{i-1}(s_{i-1} - \omega_{i-1}z_{i-1})$
6: $z_i := t_i + \beta_{i-1}(z_{i-1} - \omega_{i-1}v_{i-1})$
7: $q_i := r_i - \alpha_i s_i$
8: $y_i := w_i - \alpha_i z_i$
9: **compute** (q_i, y_i) ; (y_i, y_i)
10: $\omega_i := (q_i, y_i) / (y_i, y_i)$
11: **overlap** $v_i := Az_i$
12: $x_{i+1} := x_i + \alpha_i p_i + \omega_i q_i$
13: $r_{i+1} := q_i - \omega_i y_i$
14: $w_{i+1} := y_i - \omega_i (t_i - \alpha_i v_i)$
15: **compute** (r_0, r_{i+1}) ; (r_0, w_{i+1}) ; (r_0, s_i) ; (r_0, z_i)
16: $\beta_i := (\alpha_i/\omega_i)(r_0, r_{i+1}) / (r_0, r_i)$
17: $\alpha_{i+1} := (r_0, r_{i+1}) / ((r_0, w_{i+1}) + \beta_i (r_0, s_i) - \beta_i\omega_i (r_0, z_i))$
18: **overlap** $t_{i+1} := Aw_{i+1}$
19: **end for**
20: **end function**

dot-prod
SpMV
axpy

Pipelined BiCGStab:

(non-preconditioned)

Global communication

- 2 global red. phases (vs. 3)

Local communication

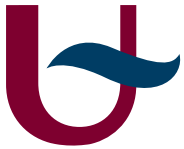
- 2 overlapping SpMVs

No communication

- 8 recurrences (vs. 4)

Status after Step 2:

both global comm. phases are overlapped with SpMV computations ('hidden'), at the cost of 4 additional axpys compared to standard BiCGStab



Bi-Conjugate Gradients Stabilized
Preconditioned pipe-BiCGStab

Algorithm 8 Preconditioned Pipelined BiCGStab

1: **function** P-PIPE-BICGSTAB(A, M^{-1}, b, x_0)
2: $r_0 := b - Ax_0$; $\hat{r}_0 := M^{-1}r_0$; $w_0 := A\hat{r}_0$; $\hat{w}_0 := M^{-1}w_0$
3: $t_0 := A\hat{w}_0$; $\alpha_0 := (r_0, r_0) \,/\, (r_0, w_0)$; $\beta_{-1} := 0$
4: **for** $i = 0, \ldots$ **do**
5: $\hat{p}_i := \hat{r}_i + \beta_{i-1}(\hat{p}_{i-1} - \omega_{i-1}\hat{s}_{i-1})$
6: $s_i := w_i + \beta_{i-1}(s_{i-1} - \omega_{i-1}z_{i-1})$
7: $\hat{s}_i := \hat{w}_i + \beta_{i-1}(\hat{s}_{i-1} - \omega_{i-1}\hat{z}_{i-1})$
8: $z_i := t_i + \beta_{i-1}(z_{i-1} - \omega_{i-1}v_{i-1})$
9: $q_i := r_i - \alpha_i s_i$
10: $\hat{q}_i := \hat{r}_i - \alpha_i \hat{s}_i$
11: $y_i := w_i - \alpha_i z_i$
12: **compute** (q_i, y_i) ; (y_i, y_i)
13: $\omega_i := (q_i, y_i) \,/\, (y_i, y_i)$
14: **overlap** $\hat{z}_i := M^{-1}z_i$
15: **overlap** $v_i := A\hat{z}_i$
16: $x_{i+1} := x_i + \alpha_i\hat{p}_i + \omega_i\hat{q}_i$
17: $r_{i+1} := q_i - \omega_i y_i$
18: $\hat{r}_{i+1} := \hat{q}_i - \omega_i(\hat{w}_i - \alpha_i\hat{z}_i)$
19: $w_{i+1} := y_i - \omega_i(t_i - \alpha_i v_i)$
20: **compute** (r_0, r_{i+1}) ; (r_0, w_{i+1}) ; (r_0, s_i) ; (r_0, z_i)
21: $\beta_i := (\alpha_i/\omega_i)\,(r_0, r_{i+1}) \,/\, (r_0, r_i)$
22: $\alpha_{i+1} := (r_0, r_{i+1}) \,/\, ((r_0, w_{i+1}) + \beta_i(r_0, s_i) - \beta_i\omega_i(r_0, z_i))$
23: **overlap** $\hat{w}_{i+1} := M^{-1}w_{i+1}$
24: **overlap** $t_{i+1} := A\hat{w}_{i+1}$
25: **end for**
26: **end function**

dot-prod
SpMV
axpy

Like for any pipelined method, including a preconditioner is easy.

Pipelined BiCGStab:

(preconditioned)

Global communication

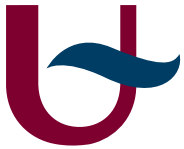
- 2 global red. phases (vs. 3)

Local communication

- 2 overlapping Prec + SpMVs

No communication

- 11 recurrences (vs. 4)



Pipe-BiCGStab vs. IBiCGStab vs. CA-BiCGStab

	GLRED	SPMV	Flops (AXPY + DOT-PROD)	Time (GLRED + SPMV)	Memory
BiCGStab	3	2	20	3 GLRED + 2 SPMV	7
IBiCGStab	1	2	30	1 GLRED + 2 SPMV	10
p-BiCGStab	2	2*	38	2 max(GLRED, SPMV)	11
s-step CA-BiCGStab	$1/s$	4	$32s$+45	$1/s$ GLRED + 4 SPMV	$4s$+5

Theoretical speed-up factors over classical BiCGStab:

	pipe-BiCGStab C., Vanroose, 2017	IBiCGStab Yang & Brent, 2002	CA-BiCGStab Carson, 2015
if time(GLRED) $\approx$ time(SPMV)	$\times$ 2.5	$\times$ 1.67	$\times$ 1.25
if time(GLRED) $\gg$ time(SPMV)	$\times$ 1.5	$\times$ 3.0	$\times$ 3s

Is algorithm with 1 GLRED overlapped with all SPMVs possible? Yes; however...

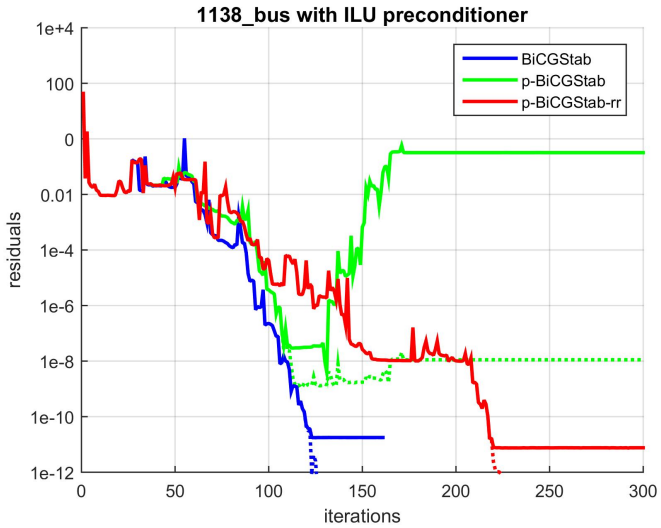
- no. axpys is much larger $\rightarrow$ algorithm robustness decreases (rounding errors)
- one extra SpMV required

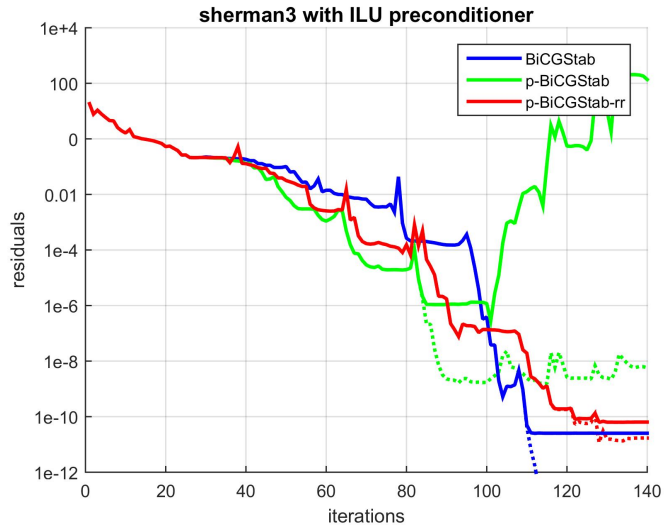
Numerical results: attainable accuracy

MatrixMarket collection: unsymmetric test problems `tol = 1e-20`

Matrix	$\|r_0\|_2$	BiCGStab		p-BiCGStab		p-BiCGStab-rr			
		iter	$\|r_i\|_2$	iter	$\|r_i\|_2$	iter	$\|r_i\|_2$	k	#nrr
1138_bus	4.3e+01	124	1.8e-11	130	4.0e-09	220	7.4e-12	35	3
add32	8.0e-03	46	7.8e-18	42	5.0e-16	51	5.7e-18	10	2
bcsstk14	2.1e+09	559	7.3e-06	444	6.6e-01	522	3.8e-03	200	2
bcsstk18	2.6e+09	523	4.8e-06	450	1.1e-01	725	2.8e-05	50	7
bcsstk26	3.5e+09	414	1.1e-05	216	5.7e-01	475	8.6e-04	30	6
bcsstm25	6.9e+07	-	3.2e+00	-	3.8e+00	-	4.6e+00	1000	9
bfw782a	3.2e-01	117	9.5e-14	106	5.1e-13	133	2.6e-15	20	4
bwm2000	1.1e+03	1733	2.5e-09	1621	1.4e-05	2231	3.8e-08	500	3
cdde6	5.8e-01	151	8.1e-14	147	2.0e-11	159	2.1e-15	10	13
fidap014	2.7e+06	-	4.3e-03	-	9.7e-03	-	4.3e-03	50	3
fs_760_3	1.6e+07	1979	1.2e-05	1039	5.1e-02	4590	1.1e-05	900	3
jagmesh9	6.8e+00	6230	2.4e-14	3582	5.8e-09	9751	1.1e-11	500	13
jpwh_991	3.8e-01	53	1.3e-14	54	1.8e-12	63	2.5e-15	10	4
orsreg_1	4.8e+00	51	4.0e-11	52	3.7e-09	56	5.9e-12	10	3
pde2961	2.9e-01	50	4.5e-15	48	3.4e-13	52	1.4e-15	10	3
rdb3200l	1.0e+01	178	3.7e-08	167	9.9e-08	181	3.4e-08	100	1
s3dkq4m2	6.8e+01	-	1.0e-05	-	1.4e-05	-	1.3e-05	1000	9
saylr4	3.1e-03	52	4.0e-12	43	7.5e-11	46	1.8e-12	10	4
sherman3	1.8e+01	111	2.5e-11	100	2.8e-07	128	6.2e-11	20	4
sstmodel	7.9e+00	-	5.1e-06	-	3.1e-06	-	4.5e-06	1000	9
utm5940	3.6e-01	256	3.0e-12	248	4.3e-08	395	2.9e-11	100	3
Average iter deviation wrt BiCGStab				-11.0%		22.1%			
Average #nrr wrt p-BiCGStab-rr iter									2.4%

Numerical results: attainable accuracy





Residual replacement every i-th iteration
(non-automated, i.e. i is a parameter of the method, but chosen large s.t. no. repl. is small)

$$r_i := b - Ax_i, \qquad \hat{r}_i := M^{-1} r_i, \qquad w_i := A\hat{r}_i,$$

$$s_i := A\hat{p}_i, \qquad \hat{s}_i := M^{-1} s_i, \qquad z_i := A\hat{s}_i.$$

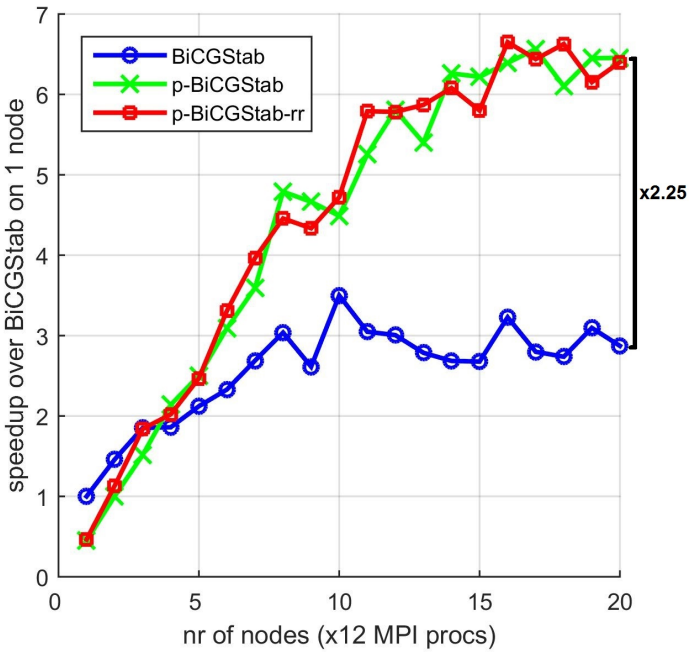
- ⊕ increased maximal attainable accuracy: comparable to BiCGStab level
- ⊕ increased robustness: counteracts rounding error amplification
- ⊖ increased number of iterations possible (convergence delay)

Numerical results: strong scaling

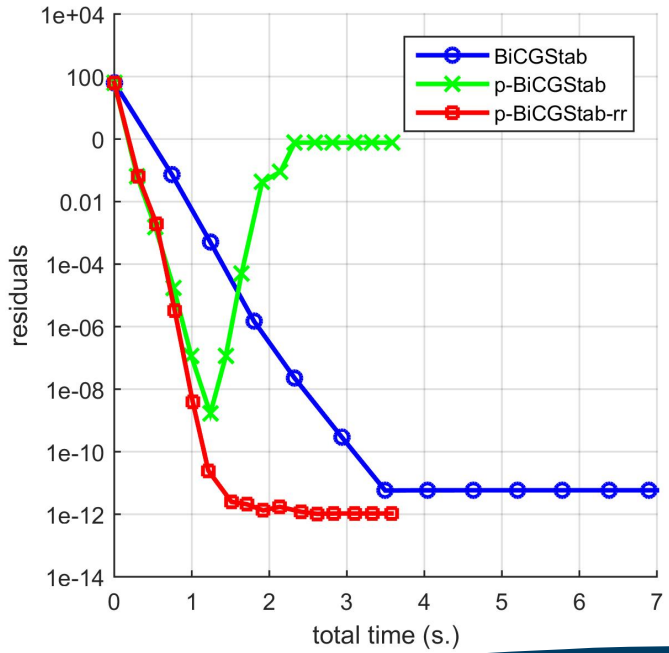
- PETSc implementation using MPICH-3.1.3 communication
- Benchmark problem 1: 2D unsymmetric model, 1,000,000 unknowns
- System specs: 20 nodes, two 6-core Intel Xeon X5660 Nehalem 2.8GHz CPUs/node

$$A_1^{stencil} = \begin{pmatrix} & 1 & \\ 1 & -4 & \varepsilon \\ & \varepsilon & \end{pmatrix}, \quad \text{with } \varepsilon = 0.999$$

Speedup over single-node BiCGStab



Accuracy i.f.o. total time spent

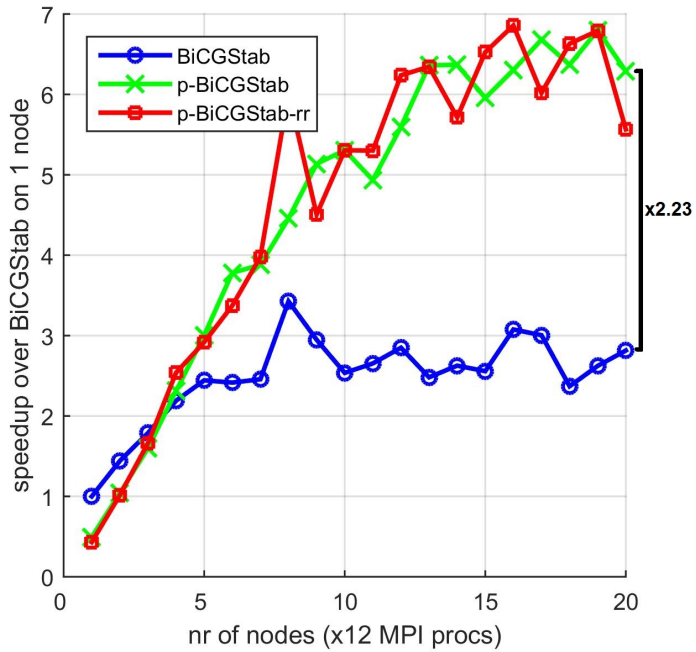


Numerical results: strong scaling

- PETSc implementation using MPICH-3.1.3 communication
- Benchmark problem 2: 2D indefinite Helmholtz model, 1,000,000 unknowns
- System specs: 20 nodes, two 6-core Intel Xeon X5660 Nehalem 2.8GHz CPUs/node

$$A_2^{stencil} = \begin{pmatrix} & 1 & \\ 1 & -1 & 1 \\ & 1 & \end{pmatrix}$$

Speedup over single-node BiCGStab



Iterations i.f.o. number of nodes

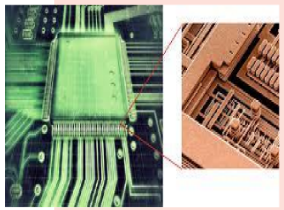
nodes	1	2	3	4	5
BiCGStab	1563	1805	1789	1875	1710
p-BiCGStab	1807	1614	1779	1787	1673
p-BiCGStab-rr	1857	1788	1728	1570	1677
nodes	6	7	8	9	10
BiCGStab	1852	1789	1555	1641	1909
p-BiCGStab	1547	1668	1773	1640	1673
p-BiCGStab-rr	1721	1688	1283	1884	1718
nodes	11	12	13	14	15
BiCGStab	1805	1715	1875	1717	1765
p-BiCGStab	1936	1811	1629	1642	1849
p-BiCGStab-rr	1845	1628	1562	1861	1650
nodes	16	17	18	19	20
BiCGStab	1722	1657	2050	1778	1670
p-BiCGStab	1843	1726	1796	1713	1870
p-BiCGStab-rr	1647	1889	1750	1701	2112

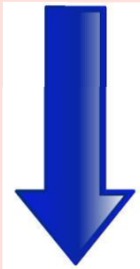
How soft errors occur?

What is soft error?

- Possible causes : electricity fluctuations, cosmic particle effects, etc...
- Appears on: memories, registers, pipeline of the processor

Exascale





SIZE OF DEVICES

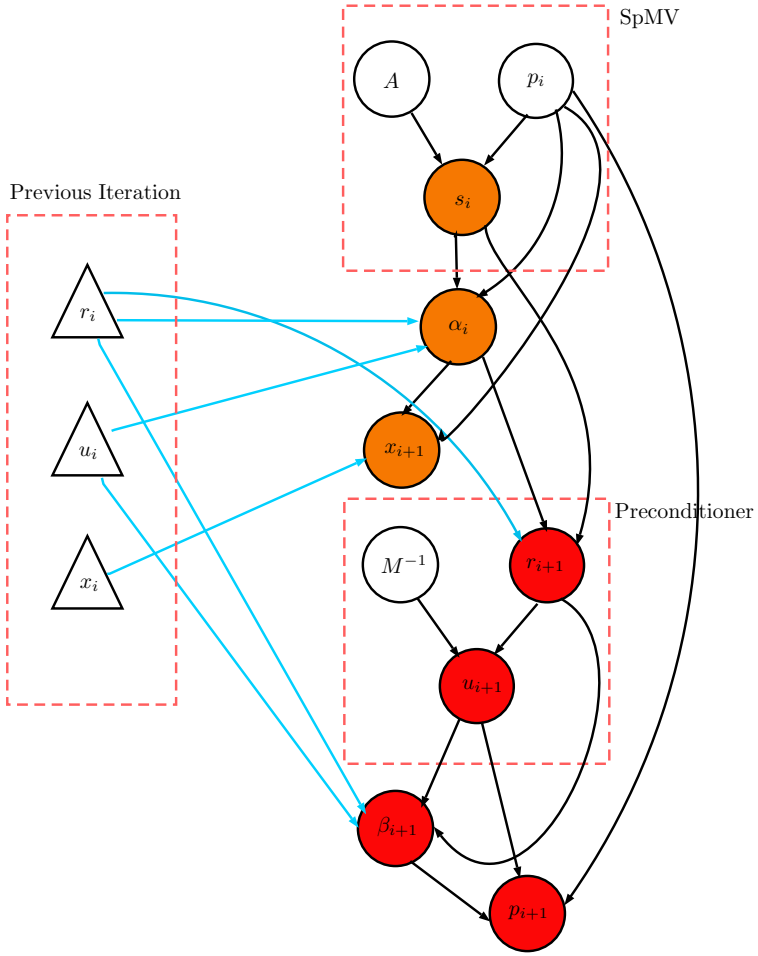
POSSIBLE SOFT ERROR RATES

OF COMPONENTS

AREA EFFECTED BY RADIATION

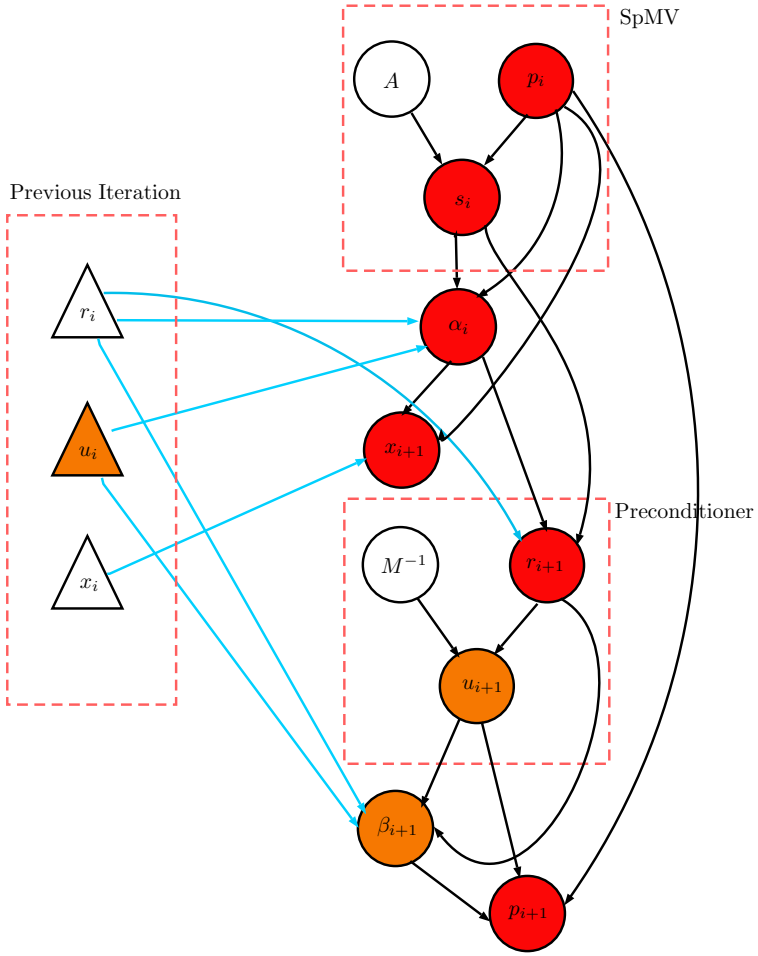
Propagation of SDC at SpMV

1: $r_0 := b - Ax_0$
2: $u_0 := M^{-1} r_0$; $p_0 := r_0$
3: **for** $i = 0, \ldots$ **do**
4: $\quad s := Ap_i$
5: $\quad \alpha := (r_i, u_i)/(s, p_i)$
6: $\quad x_{i+1} := x_i + \alpha p_i$
7: $\quad r_{i+1} := r_i - \alpha s$
8: $\quad u_{i+1} := M^{-1} r_{i+1}$
9: $\quad \beta := (r_{i+1}, u_{i+1})/(r_i, u_i)$
10: $\quad p_{i+1} := u_{i+1} + \beta p_i$
11: **end for**



Propagation of SDC at Preconditioner

1: $r_0 := b - Ax_0$
2: $u_0 := M^{-1} r_0$; $p_0 := r_0$
3: **for** $i = 0, \ldots$ **do**
4: $\quad s := Ap_i$
5: $\quad \alpha := (r_i, u_i)/(s, p_i)$
6: $\quad x_{i+1} := x_i + \alpha p_i$
7: $\quad r_{i+1} := r_i - \alpha s$
8: $\quad u_{i+1} := M^{-1} r_{i+1}$
9: $\quad \beta := (r_{i+1}, u_{i+1})/(r_i, u_i)$
10: $\quad p_{i+1} := u_{i+1} + \beta p_i$
11: **end for**



Aim of this study

Context

Soft Errors in Preconditioned Conjugate Gradient (PCG)

Question 1

Impact of soft errors on convergence of PCG

Question 2

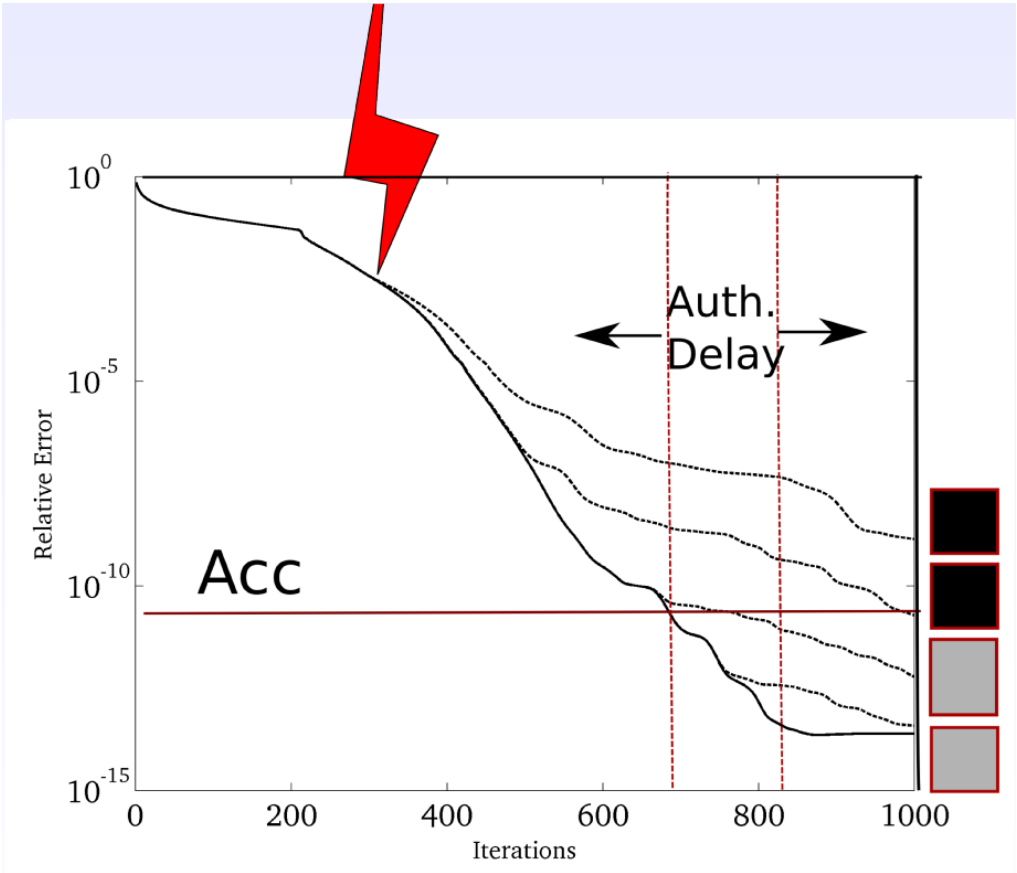
Reliability of numerical detection mechanisms ?

Question 3

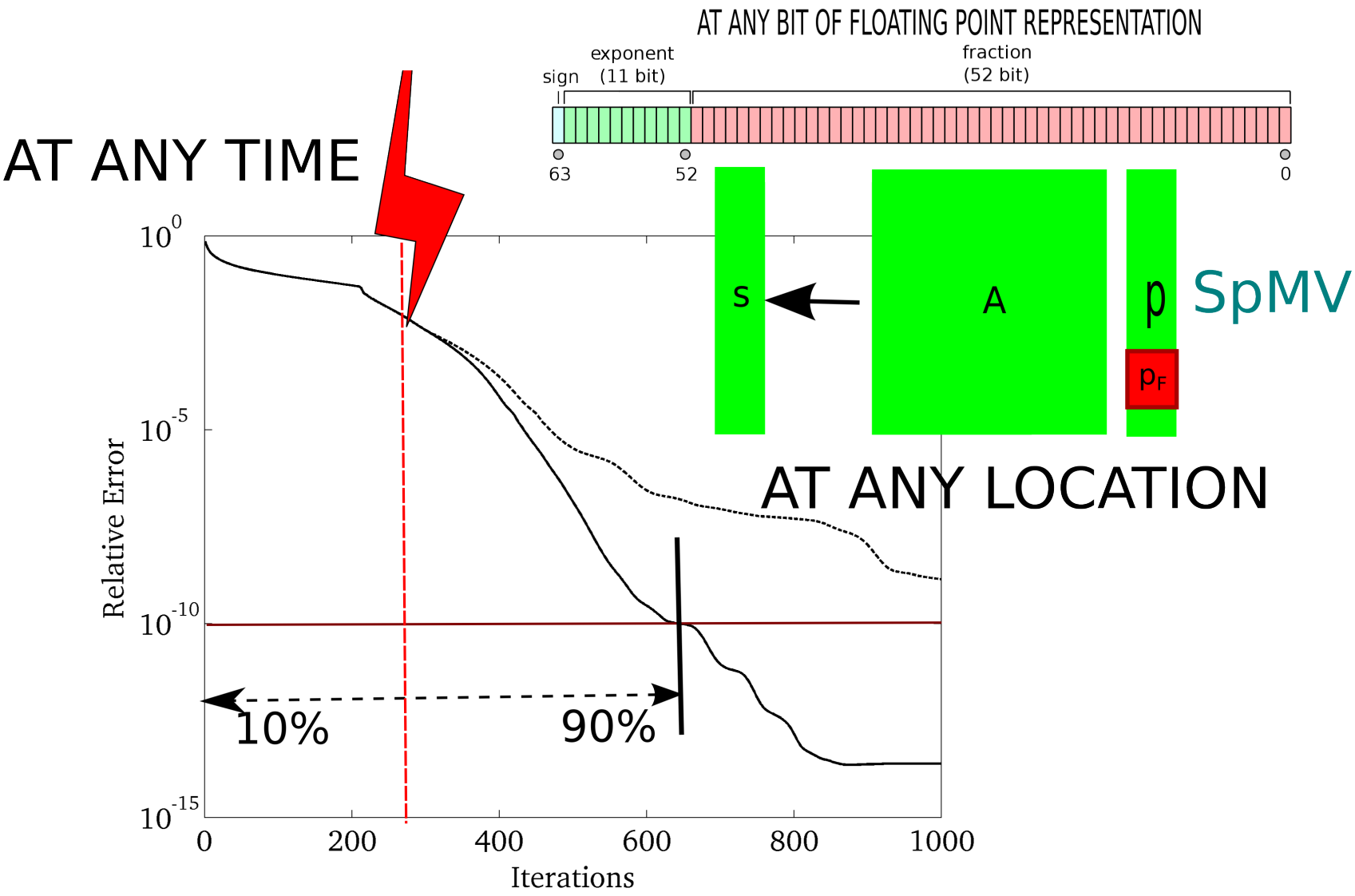
Robust numerical recovery schemes ?

Protocol for sensitivity study

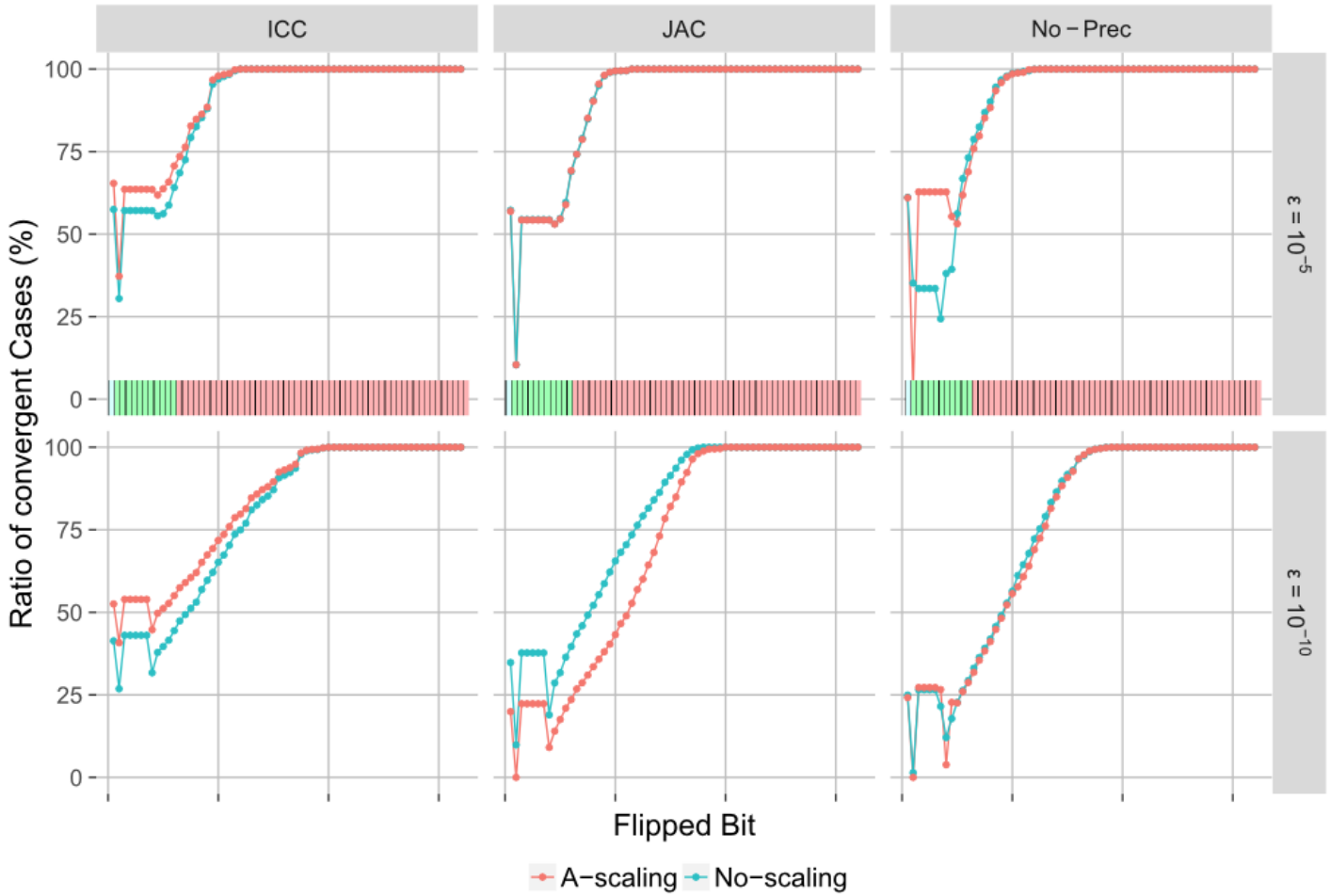
	Effect on the convergence
Converged	
Not Converged	



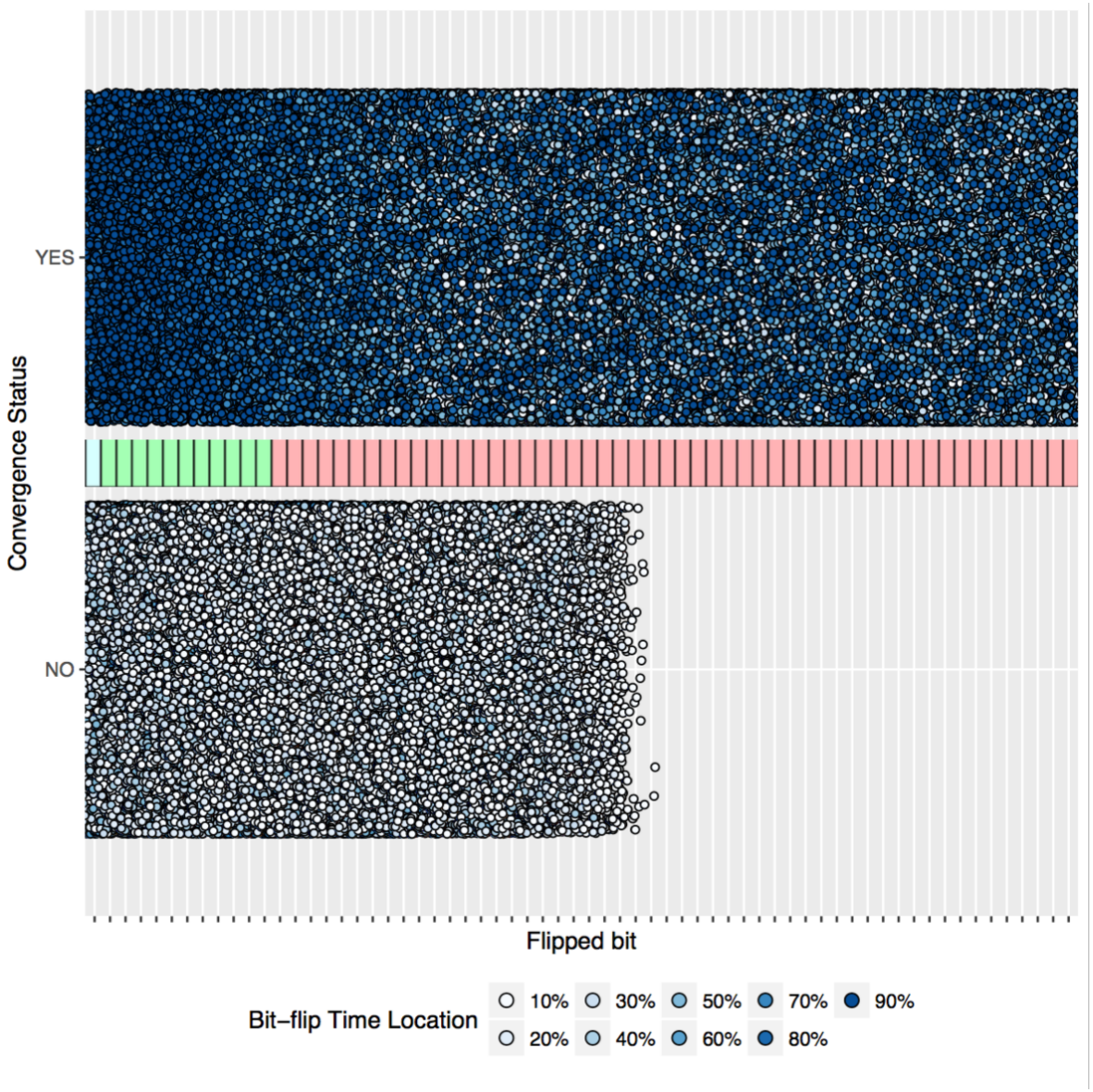
Fault injection methodology in 64-bit



Sensitivity v.s. SpMV bit-flip locations



A closer look at temporal behaviour of SpMV soft-errors



Check the impact of SpMV computation

Rounding error analysis of residuals in finite precision arithmetics

Residual Deviation

- True Residual in EXACT: $\hat{r}_{i+1} := b - Ax_{i+1}$
- Computed residual in EXACT: $r_{i+1} := r_i - \alpha s$
- Equality in EXACT $\Delta^r_{i+1} = \hat{r}_{i+1} - r_{i+1} = 0$

Bound for deviation [Vorst & Yee, SISC, 2000]

$$||\Delta^r_{i+1}|| \leq \mathbf{u}N||A|| \sum_{j=0}^{i} ||x_j|| + \mathbf{u} \sum_{j=0}^{i} ||r_j||$$

- $\mathbf{u}$: Machine epsilon
- N : Maximum nnz per row

Check the impact of SpMV computation

Rounding error analysis of residuals in finite precision arithmetics

Residual Deviation

- True residual in FINITE: $\hat{r}_{i+1} := b - A(x_i + \alpha_i p_i + \delta_i^x)$
- Computed residual in FINITE: $r_{i+1} := r_i - \alpha_i s_i + \delta_i^r$
- Deviation in FINITE:
  $\Delta_{i+1}^r := \Delta_i^r + A\alpha_i p_i - \alpha_i A p_i + A\delta_i^x + \delta_i^r$

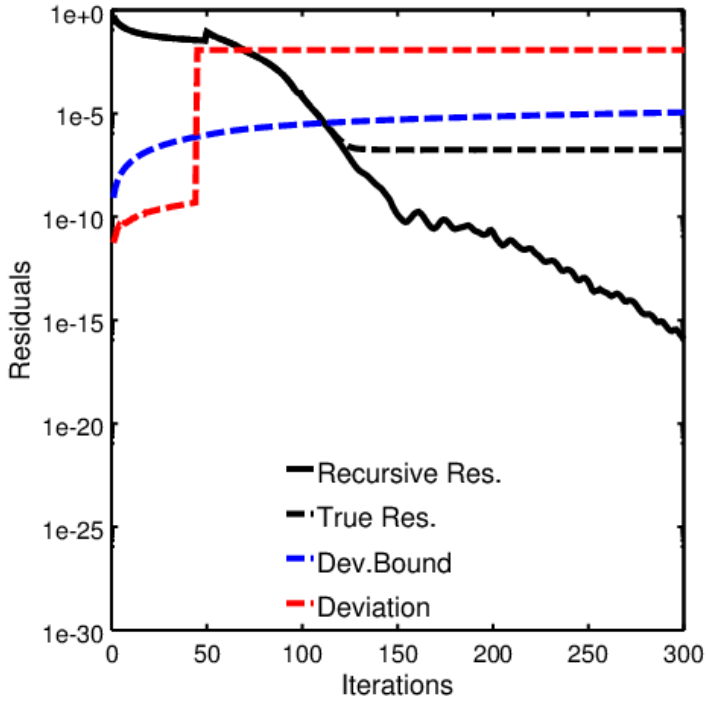
Bound for deviation [Vorst & Yee, SISC, 2000]

$||\Delta_{i+1}^r|| \leq \mathbf{u}N||A|| \sum_{j=0}^{i} ||x_j|| + \mathbf{u} \sum_{j=0}^{i} ||r_j||$

- $\mathbf{u}$: Machine epsilon
- N : Maximum nnz per row

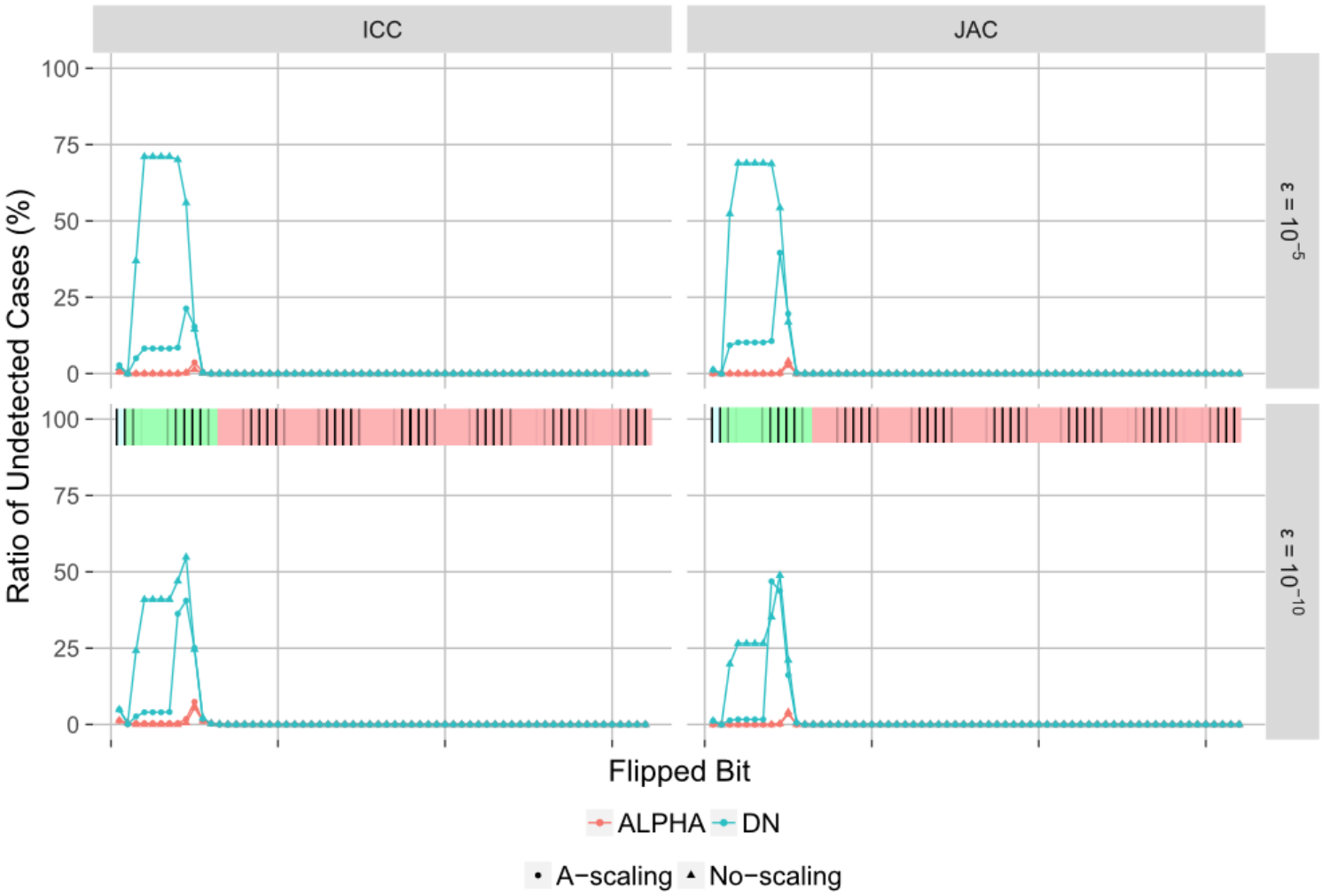
Check the impact of SpMV computation

How can we employ this bound to detect soft errors?

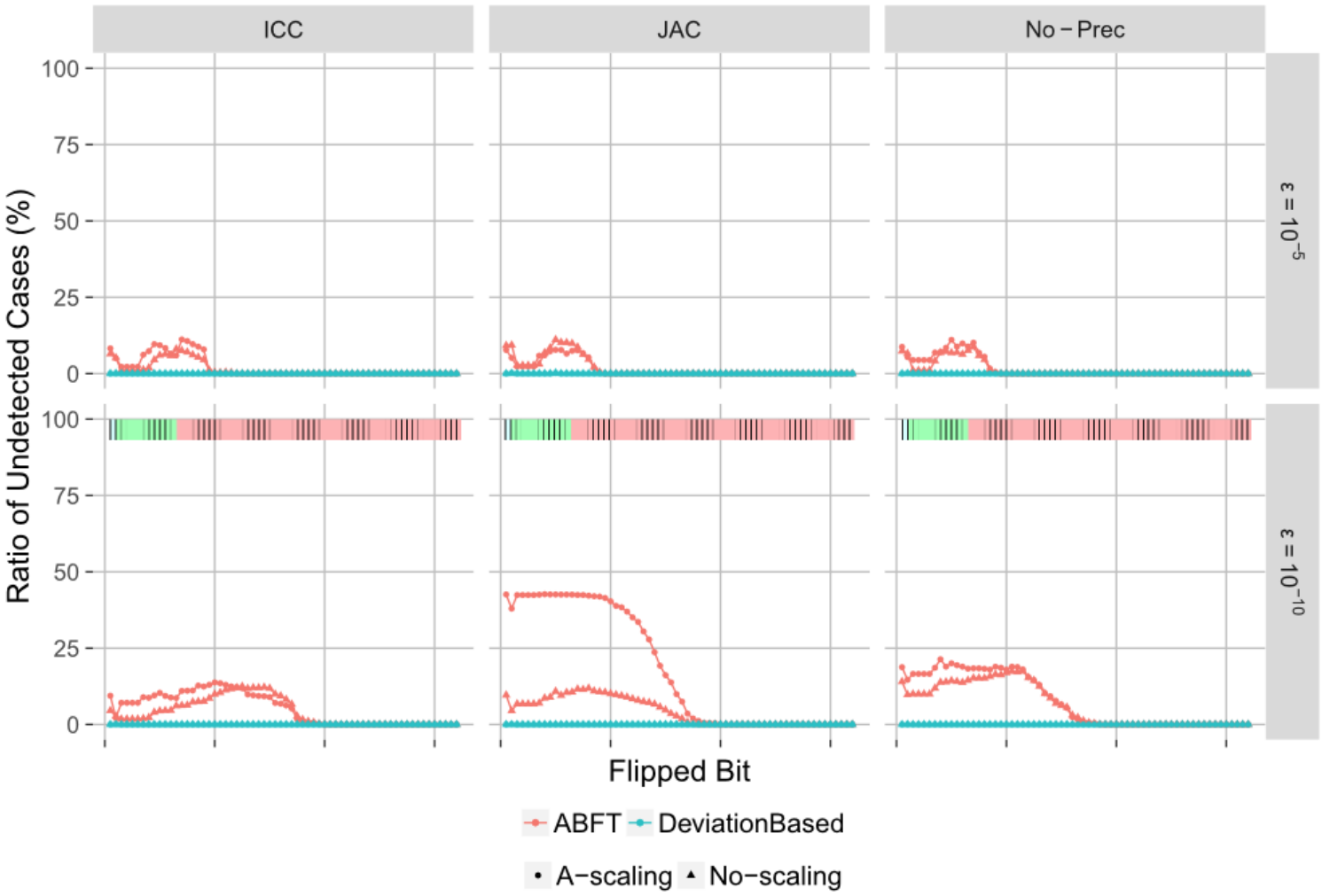


- Soft errors at SpMV will effect x_{i+1} and r_{i+1} in a different way
- It will create a deviation larger than the theoretical upper bound
- It gives us an opportunity to detect the soft errors

Detection robustness v.s. Preco. bit-flip location



Detection robustness v.s. SpMV bit-flip location



Hybrid approach for full-protection

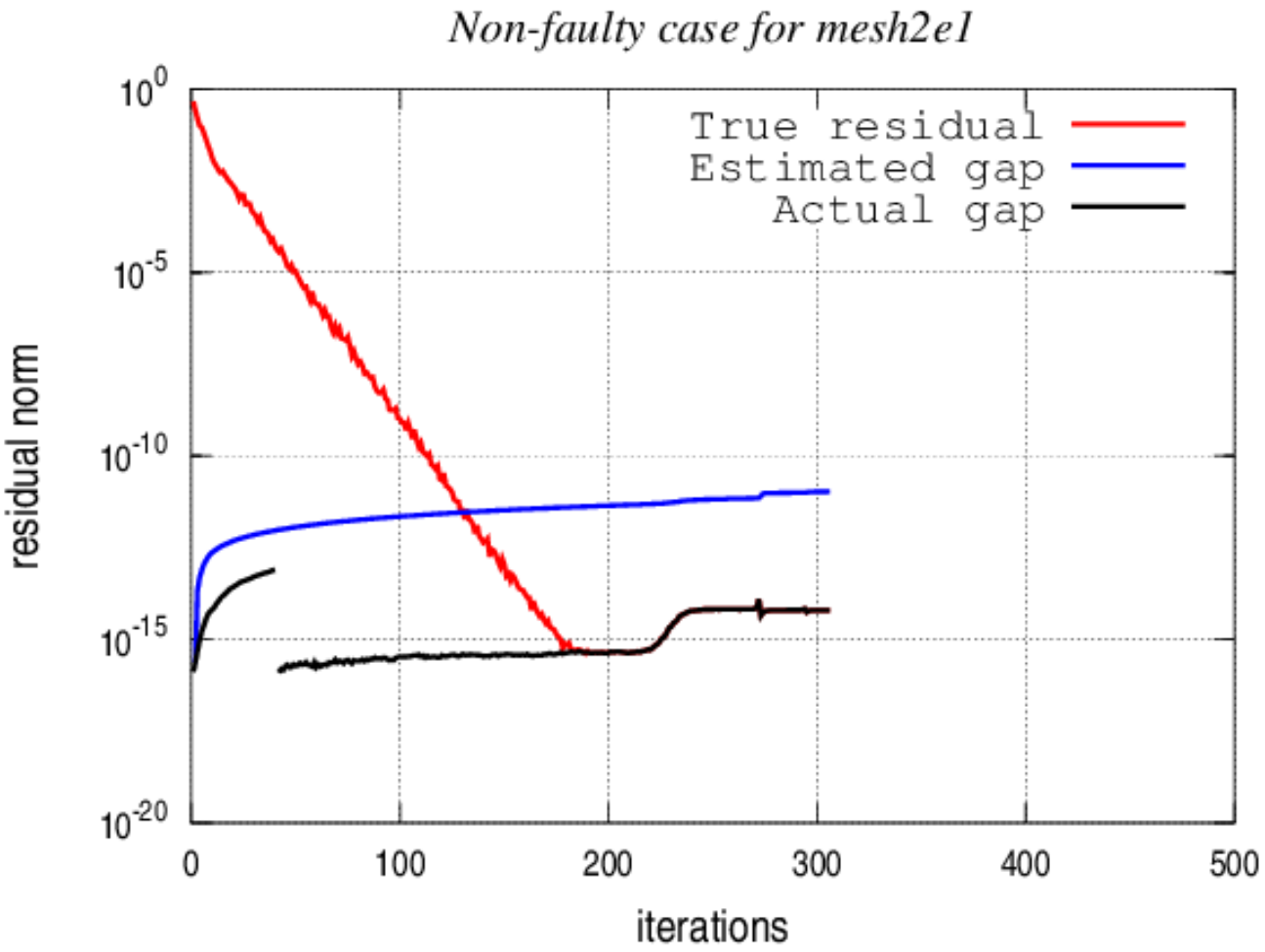
Level-1 : Deviation based approach to detect SpMV faults

Deviation can be checked periodically to avoid extra SpMV at each iteration

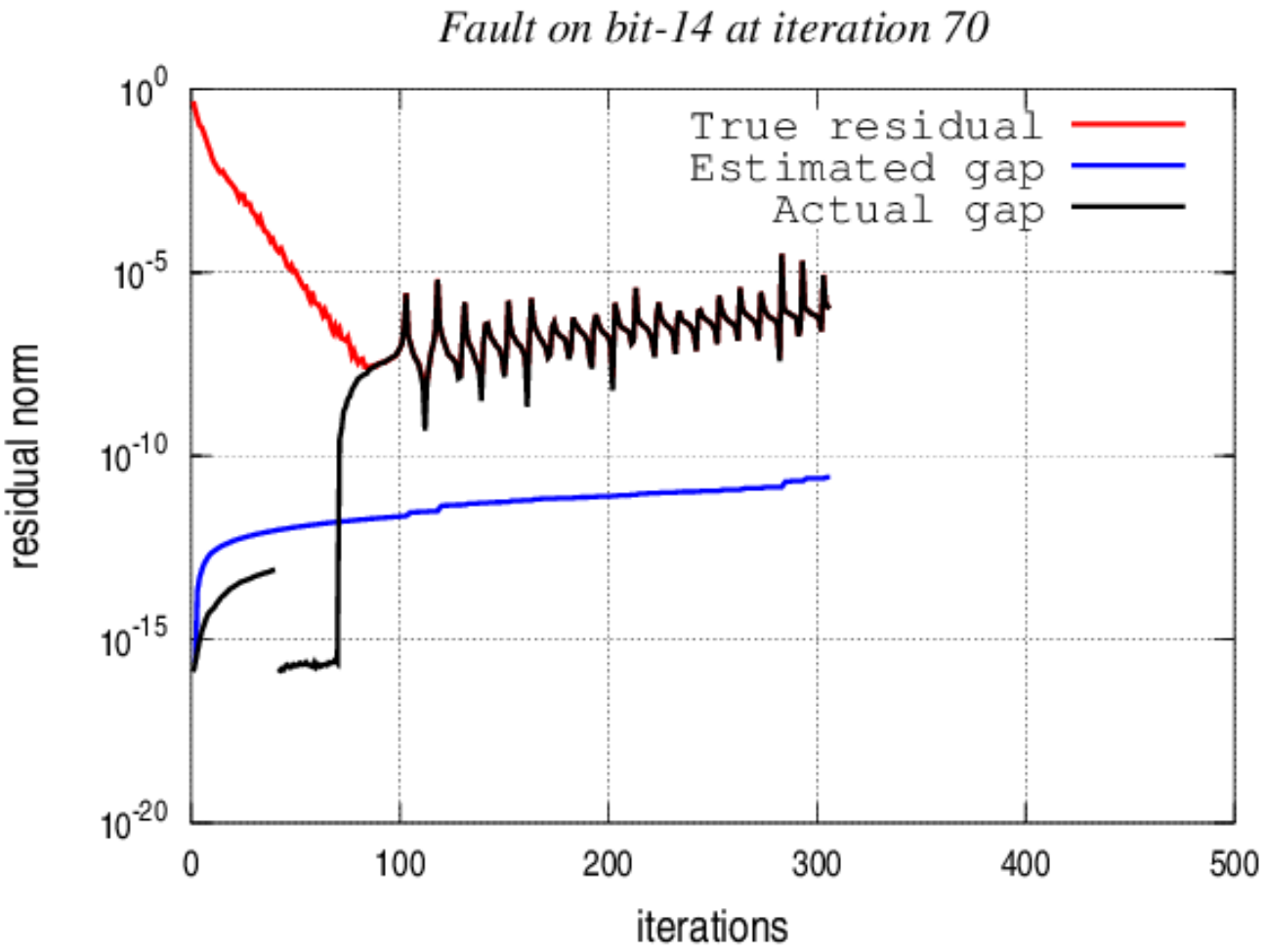
Level-2 : α based approach to detect preconditioner faults

An estimation for λ_{max} for preconditioned matrix is required

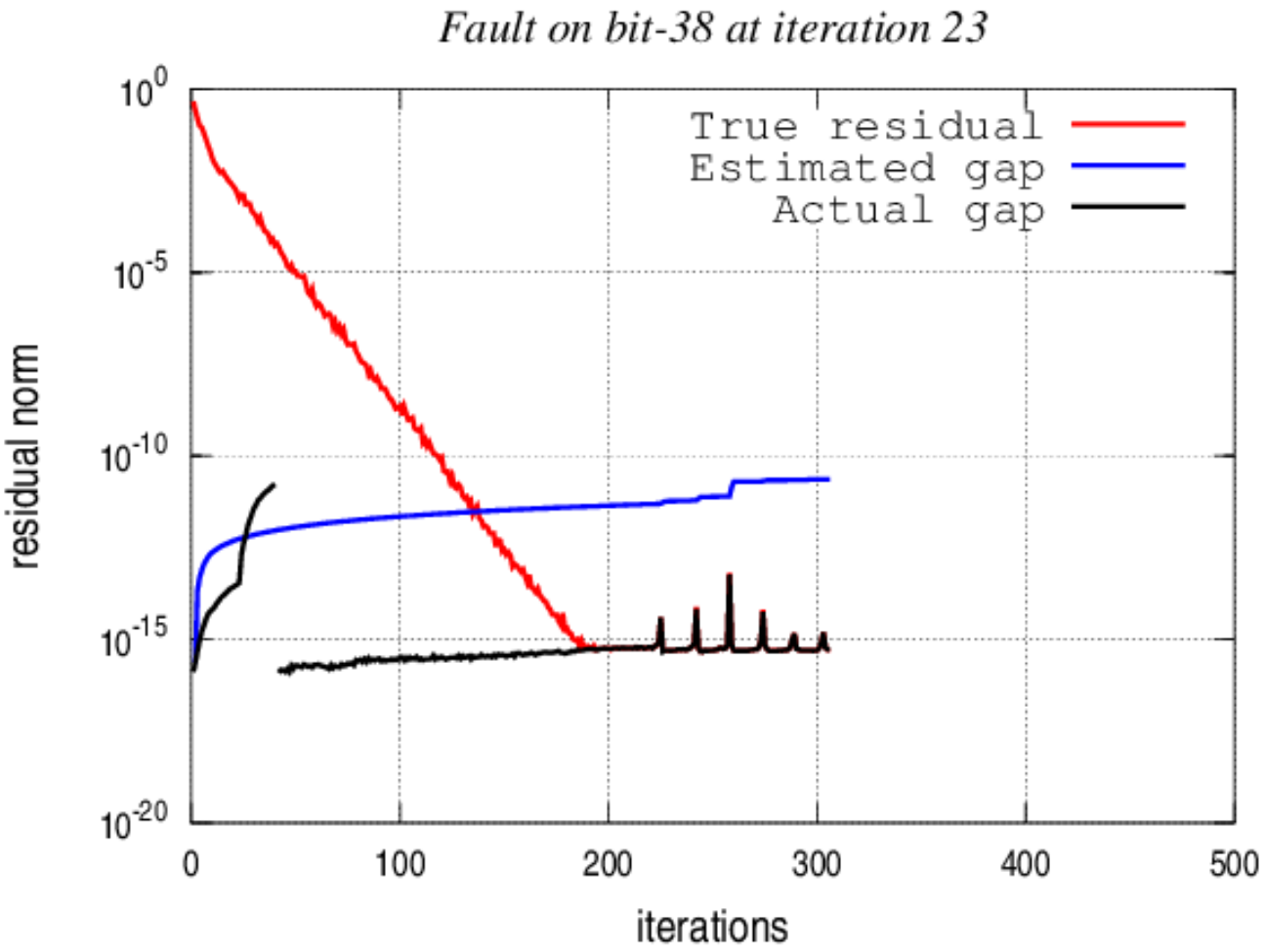
Non-faulty case for matrix mesh2e1



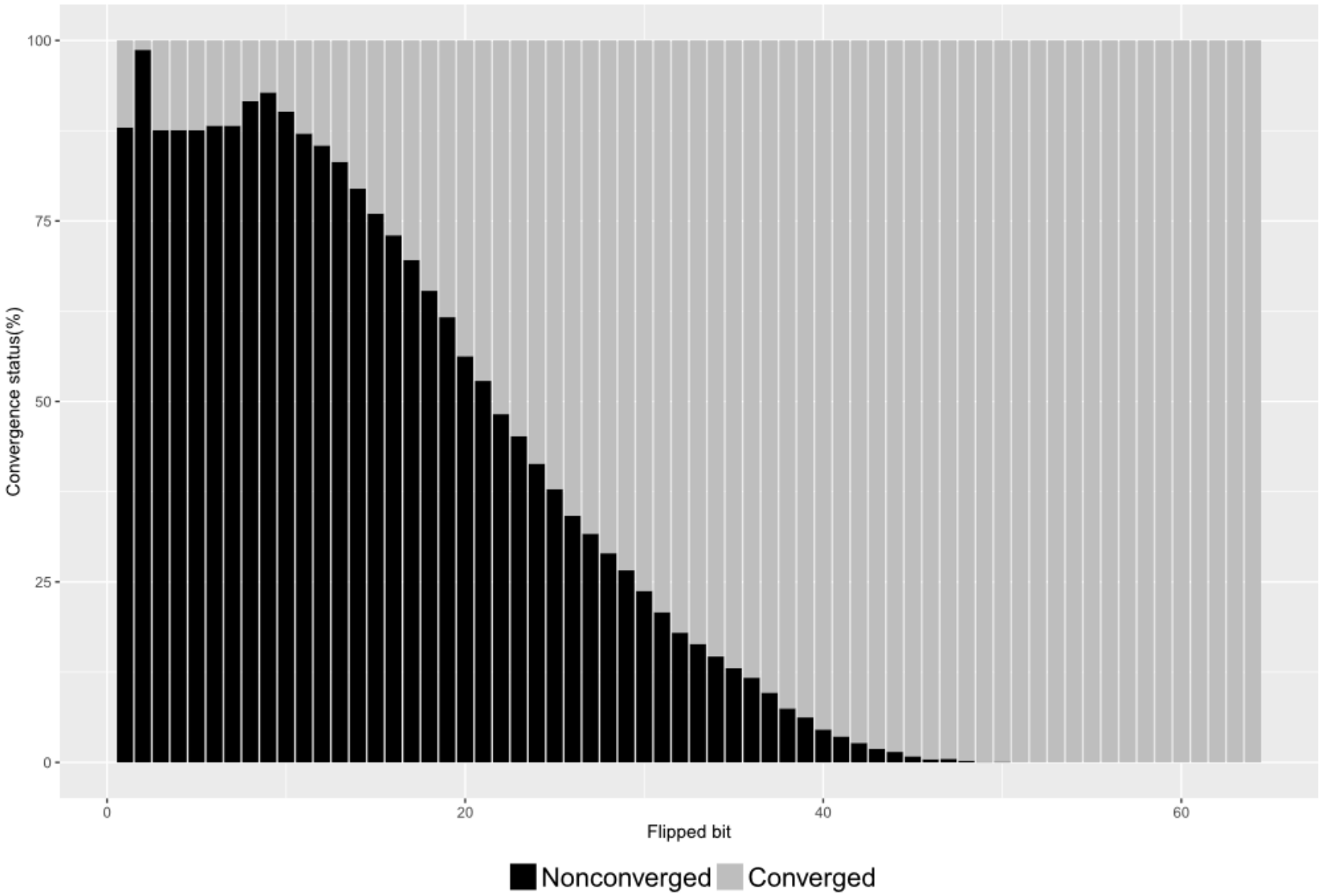
Several Bit-flip effects for matrix mesh2e1



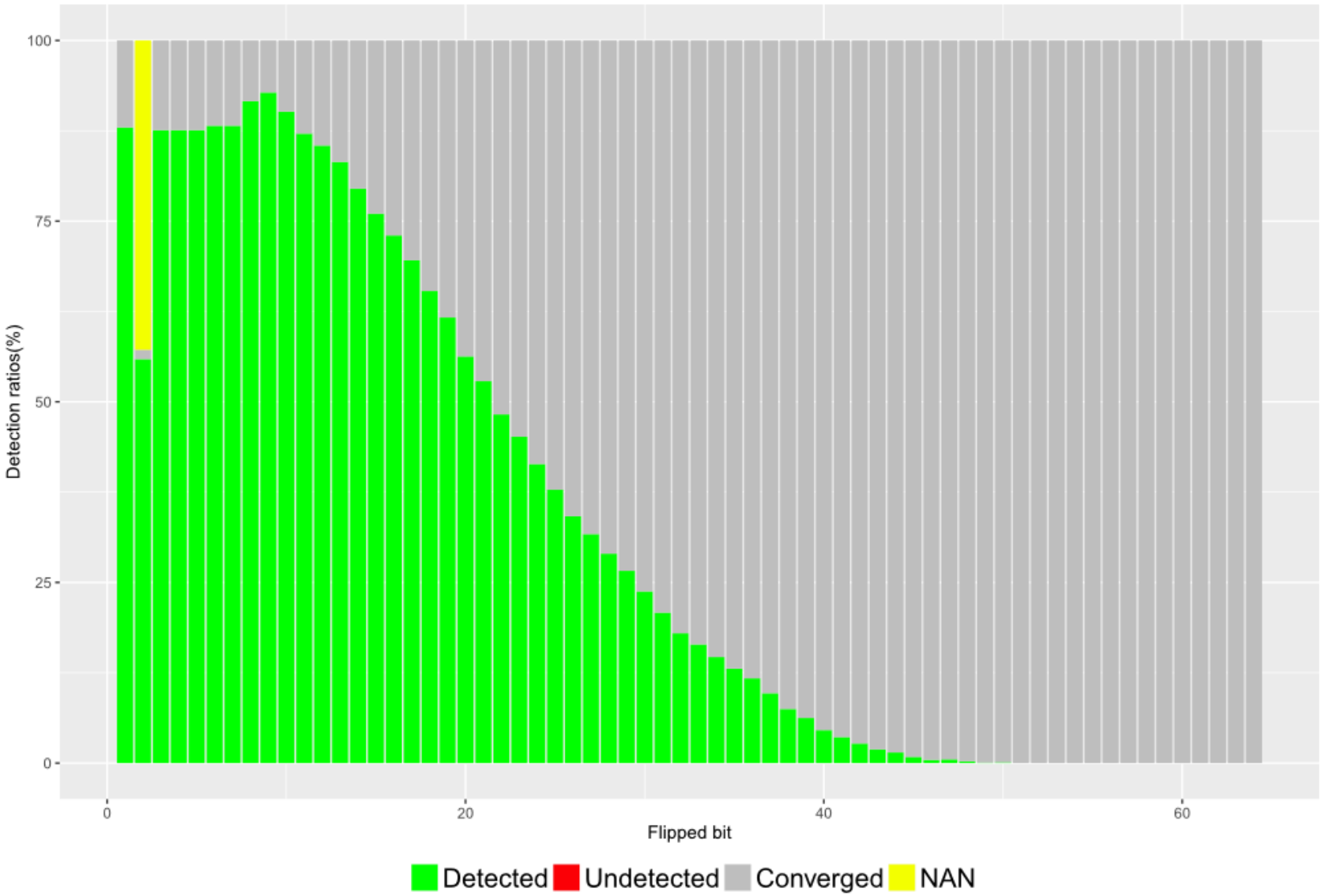
Several Bit-flip effects for matrix mesh2e1



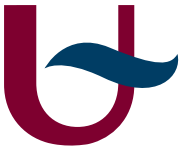
Sensitivity behaviour of all matrices - 200.000 experiments



Detection success for all matrices - 200.000 experiments



¿



Overview

Analysis of rounding error propagation in pipelined CG

- pipe-CG is much more sensitive to rounding errors than standard CG
- rounding error model allows for real-time countermeasures: pipe-CG-rr

Shifted pipelined CG

- introduce shift σ in auxiliary variables: $s_i = (A - \sigma)p_i$, $w_i = (A - \sigma)s_i$
- stabilizing effect on rounding error propagation for the right shift choice

Longer pipelines: pipe(ℓ)-CG

- useful when time(GLRED) is much larger than time(SPMV+PREC)
- further reduced numerical stability (WIP)

Pipelined BiCGStab

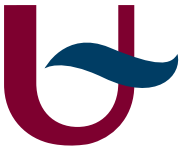
- pipe-BiCGStab for non-symmetric operators (alternative to pipe-GMRES)
- implement countermeasures to improve rounding error resilience

Soft error detection

- rounding error model allows for separation rounding errors vs. soft errors
- detect-and-correct strategy to improve soft error resilience

References

- P. Ghysels, T.J. Ashby, K. Meerbergen, W. Vanroose, *Hiding Global Communication Latency in the GMRES Algorithm on Massively Parallel Machines*, SIAM J. Sci. Comput., 35(1), 2013.
- P. Ghysels, W. Vanroose, *Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm*, Parallel Computing, 40, 2013.
- H.A. Van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, 13(2):631–644, 1992.
- H.A. Van der Vorst, Q. Ye, *Residual Replacement strategies for Krylov Subspace iterative methods for Convergence of the True Residuals*, 22(3), SIAM J. Sci. Comput., 2000.
- E. Carson, J. Demmel, *A Residual Replacement Strategy for Improving the Maximum Attainable Accuracy of s-Step Krylov Methods*, SIAM J. Mat. Anal. Appl., 35(1), 2014.
- S. Cools, W. Vanroose, *Analyzing the effect of local rounding error propagation on the maximal attainable accuracy of the pipelined Conjugate Gradients method*, under review, SIAM J. Numer. Anal., 2017.
- S. Cools, W. Vanroose, *The communication-hiding pipelined BiCGstab method for the parallel solution of large unsymmetric linear systems*, Parallel Computing, 65, pp.1–20, 2017.
- S. Cools, W. Vanroose, *Numerically Stable Variants of the Communication-hiding Pipelined Conjugate Gradients Algorithm for the Parallel Solution of Large Scale Symmetric Linear Systems*, Technical Report, `https://arxiv.org/abs/1706.05988`, 2017.



Thank you for your attention!

Q: *What is the difference between pipelined and s-step Krylov methods?*
Carson, Knight, Demmel, 2013

A: Global communication is hidden vs. avoided

A: Off-the-shelf preconditioning possible vs. specialized preconditioning required

Q: *Is the code available online?*

A: Pipe-CG, pipe-CG-rr, gropp-CG, pipe-GMRES and pipe-BiCGStab are included in the current PETSc version, available at `https://www.mcs.anl.gov/petsc/`.

A: The inclusion of pipe(ℓ)-CG in PETSc is work in progress.