



## Hiding Global Reduction Latency in Pipelined Krylov Methods

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

17th GAMM Workshop, University of Cologne, Germany, September 7-8, 2017

University of Antwerp\* [BE] & INRIA Bordeaux<sup>†</sup> [FR]

Siegfried Cools\*, J. Cornelis\*, W. Vanroose\*, E. F. Yetkin<sup>†</sup>, E. Agullo<sup>†</sup>, L. Giraud<sup>†</sup>

Contact: [siegfried.cools@uantwerp.be](mailto:siegfried.cools@uantwerp.be)



# Introduction

## What are we working on?

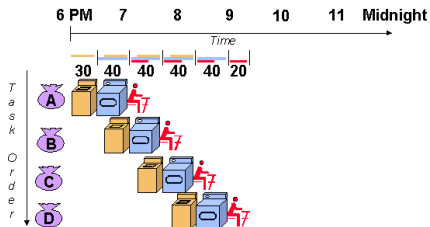
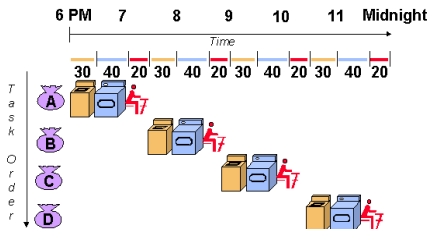
### Classical Krylov subspace method

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

vs.

### Pipelined Krylov subspace method

*Latency hiding of global drying in pipelined 'Laundry method'*





**Work initiated under:**

*EXascale Algorithms and Advanced Computational Techniques*

<http://exa2ct.eu/>

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

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



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

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

- ▶ 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:
  - dot-product
  - SpMV
  - 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  $i = 0, \dots$  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



## Krylov subspace methods

### Classical CG

#### Algorithm 1 CG

```
1: procedure CG( $A, b, x_0$ )
2:    $r_0 := b - Ax_0$ ;  $p_0 := r_0$ 
3:   for  $i = 0, \dots$  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

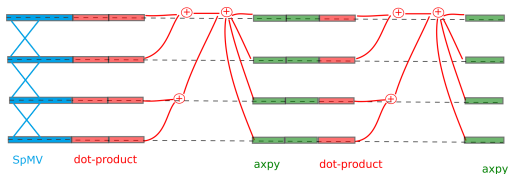
- ▶ 2 global reduction phases
- ▶ latency dominated
- ▶ scales as  $\log_2(\#partitions)$

#### 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.)





## Krylov subspace methods Pipelined CG

### Algorithm 3 Pipelined CG

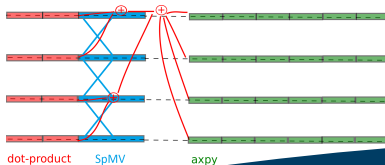
```
1: procedure PIPE-CG( $A, b, x_0$ )
2:    $r_0 := b - Ax_0; w_0 := Ar_0$ 
3:   for  $i = 0, \dots$  do
4:      $\gamma_i := (r_i, r_i)$ 
5:      $\delta := (w_i, r_i)$ 
6:      $q_i := Aw_i$ 
7:     if  $i > 0$  then
8:        $\beta_i := \gamma_i / \gamma_{i-1}; \alpha_i := (\delta / \gamma_i - \beta_i / \alpha_{i-1})^{-1}$ 
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

### 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  $s_j = Ap_j$ ,  
 $w_j = Ar_j$ ,  $z_j = As_j$  (line 12+13+17)

 Ghysels & Vanroose (2013)





## Bi-Conjugate Gradients Stabilized BiCGStab

---

### Algorithm 4 Standard BiCGStab

---

```
1: function BICGSTAB( $A, b, x_0$ )
2:    $r_0 := b - Ax_0$ ;  $p_0 := r_0$ 
3:   for  $i = 0, \dots$  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

### 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*



## Bi-Conjugate Gradients Stabilized Step 1. Avoiding communication

---

### Algorithm 4 Standard BiCGStab

---

```
1: function BICGSTAB( $A, b, x_0$ )
2:    $r_0 := b - Ax_0$ ;  $p_0 := r_0$ 
3:   for  $i = 0, \dots$  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)





## Bi-Conjugate Gradients Stabilized 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  $i = 0, \dots$  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:      $q_i := r_i - \alpha_i s_i$ 
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:     $x_{i+1} := x_i + \alpha_i p_i + \omega_i q_i$ 
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
```

dot-prod  
SpMV  
axpy

### Communication-avoiding BiCGStab: (non-preconditioned)

#### Global communication

- ▶ 2 global red. phases (vs. 3)

#### Local communication

- ▶ 2 non-overlapping SpMV's

#### No communication

- ▶ 6 recurrences (vs. 4)

#### Status after Step 1:

*number of global comm. phases reduced from 3 to 2, at the cost of 2 axpys*

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



## Bi-Conjugate Gradients Stabilized Step 2. Hiding 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  $i = 0, \dots$  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:      $q_i := r_i - \alpha_i s_i$ 
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:     $x_{i+1} := x_i + \alpha_i p_i + \omega_i q_i$ 
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
```

dot-prod  
SpMV  
axpy

(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



## Bi-Conjugate Gradients Stabilized 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  $i = 0, \dots$  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  $i = 0, \dots$  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:     $\hat{r}_{i+1} := \hat{q}_i - \omega_i (\hat{w}_i - \alpha_i z_i)$ 
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 - in theory - easy.*

### Pipelined BiCGStab:

(preconditioned)

### Global communication

- ▶ 2 global red. phases (vs. 3)

### Local communication

- ▶ 2 overlapping Prec + SpMVs

### No communication

- ▶ 11 recurrences (vs. 4)



# Bi-Conjugate Gradients Stabilized

## Pipe-BiCGStab vs. IBiCGStab vs. s-step 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 $\text{time}(\text{GLRED}) \approx \text{time}(\text{SPMV})$	$\times 2.5$	$\times 1.67$	$\times 1.25$
if $\text{time}(\text{GLRED}) \gg \text{time}(\text{SPMV})$	$\times 1.5$	$\times 3.0$	$\times 3s$

*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



## Krylov subspace methods

### Other pipelined Krylov methods

#### ▶ Pipelined GMRES

 Ghysels et al. (2012)

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

- compute  $\ell$  new basis vectors for Krylov subspace (SpMV) during global communication (dot-products).
- deeper/variable pipelining possible:  $p(\ell)$ -GMRES

see talk by **W. Vanroose**

#### ▶ Pipelined CG

 Ghysels et al. (2013)

- deeper pipelining also possible:  $p(\ell)$ -CG

see talk by **W. Vanroose**

#### ▶ Pipelined BiCGStab

 C., Vanroose (2017)

- non-symmetric operators

#### ▶ Preconditioned pipelined variants are available

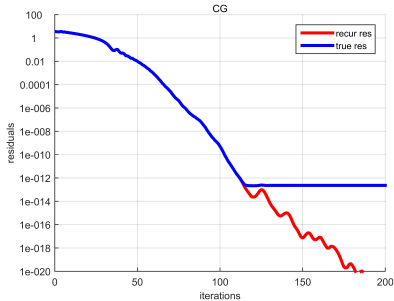
- prec-pipe-CG
- prec-pipe-GMRES
- prec-pipe-BiCGStab

#### ▶ Augmented and deflated pipelined methods are available

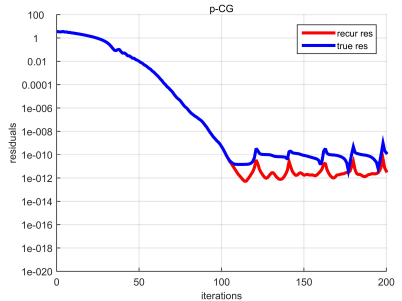


# Conjugate Gradients Rounding error propagation

## Classical CG

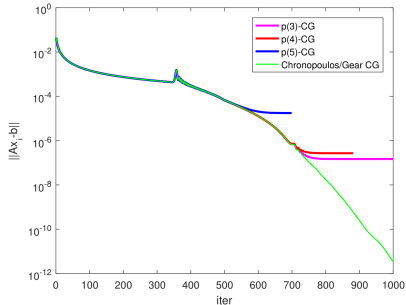


## Pipelined CG



*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



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

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





## Conjugate Gradients 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:

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

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



## Conjugate Gradients 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.

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

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

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

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

**Amplification** of local rounding errors possible, depending on  $\alpha_i$ 's and  $\beta_i$ 's.



## Conjugate Gradients Rounding error model for CG

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

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

### In practice:

- ▶ *characterizes extreme case rounding error effects*
- ▶ *often largely overestimates the actual errors*

**Error estimates:** Local rounding errors can be approximated as

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

### In practice:

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



## Conjugate Gradients 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:

$$\begin{aligned}\bar{r}_{i+1} &= \text{fl}(b - A\bar{x}_{i+1}), & \bar{w}_{i+1} &= \text{fl}(A\bar{u}_{i+1}), \\ \bar{s}_i &= \text{fl}(A\bar{p}_i), & \bar{z}_i &= \text{fl}(A\bar{q}_i).\end{aligned}$$

**Residual replacement criterion:**

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

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

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

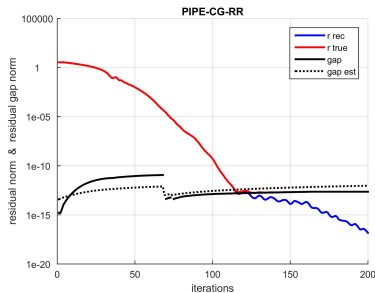
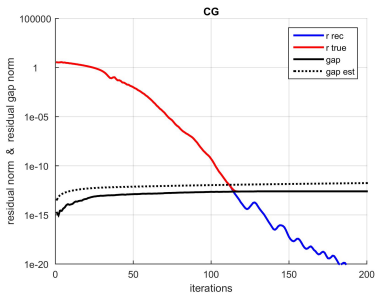
*Estimate for gap  $\|\bar{f}_i\|$  can be computed at runtime (without additional overhead), so **fully automated replacement strategy** is possible for pipe-CG.*



# Conjugate Gradients

## Pipe-CG with automated residual replacement

Example: 2D Laplacian - 2,500 unk (cont.)



- ▶ 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



# Conjugate Gradients

## Numerical results: attainable accuracy

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

Matrix	Prec	$\kappa(A)$	$n$	# <i>nnz</i>	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

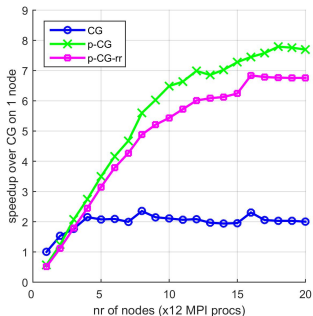


# Conjugate Gradients

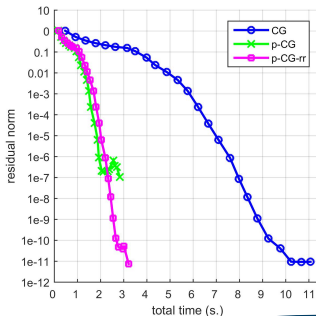
## Numerical results: strong scaling

- ▶ 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)

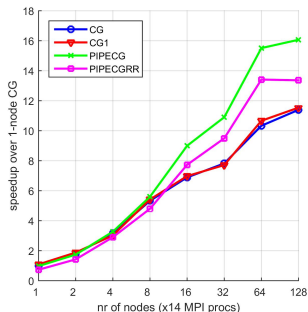




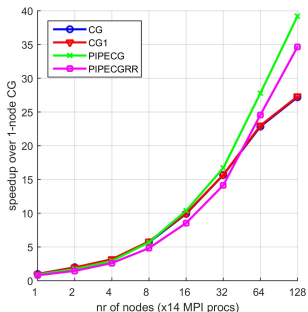
## Conjugate Gradients Numerical results: strong scaling

- ▶ 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)



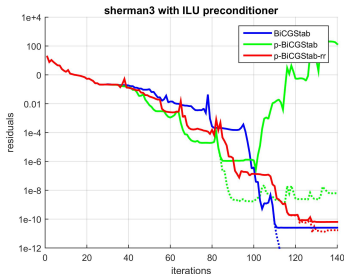
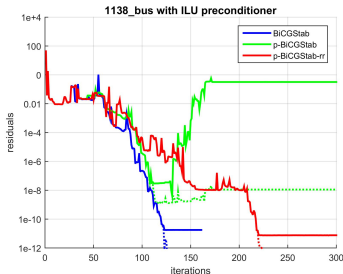
Speedup over single-node CG  
(12,500,000 unk)







## Bi-Conjugate Gradients Stabilized Numerical results: attainable accuracy



*Non-automated residual replacement every  $i$ -th iteration:*

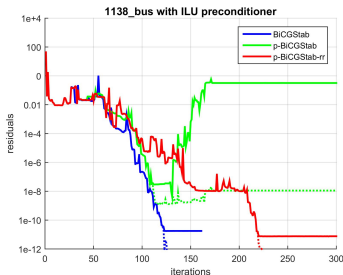
$$r_i := b - Ax_i, \quad \hat{r}_i := M^{-1}r_i, \quad w_i := A\hat{r}_i, \quad s_i := A\hat{p}_i, \quad \hat{s}_i := M^{-1}s_i, \quad z_i := A\hat{s}_i.$$

- ⊕ increased **maximal attainable accuracy**: comparable to BiCGStab level
- ⊕ increased **robustness**: resets accumulated rounding errors
- ⊖ **delay of convergence**: increased number of iterations possible

 Strakos & Tichy, 2002



## Bi-Conjugate Gradients Stabilized Numerical results: attainable accuracy



### Multi-term auxiliary recurrences



### Local error propagation matrix

$$\begin{bmatrix} 1 & -\bar{\alpha}_i \bar{\beta}_i & -\bar{\alpha}_i - \bar{\omega}_i & \bar{\alpha}_i \bar{\beta}_i (\bar{\omega}_i + \bar{\omega}_{i-1}) \\ 0 & \bar{\beta}_i & 1 & -\bar{\beta}_i \bar{\omega}_{i-1} \\ 0 & 0 & 1 & -\bar{\alpha}_i \bar{\beta}_i \\ 0 & 0 & 0 & \bar{\beta}_i \end{bmatrix}$$

*Non-automated residual replacement every  $i$ -th iteration:*

$$r_i := b - Ax_i, \quad \hat{r}_i := M^{-1} r_i, \quad w_i := A \hat{r}_i, \quad s_i := A \hat{p}_i, \quad \hat{s}_i := M^{-1} s_i, \quad z_i := A \hat{s}_i.$$

- ⊕ increased **maximal attainable accuracy**: comparable to BiCGStab level
- ⊕ increased **robustness**: resets accumulated rounding errors
- ⊖ **delay of convergence**: increased number of iterations possible

Strakos & Tichy, 2002

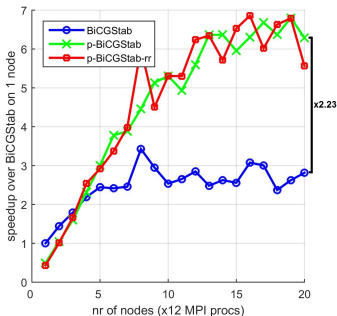


## Bi-Conjugate Gradients Stabilized Numerical results: strong scaling

- ▶ 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



## In this talk

---

### Pipelined Conjugate Gradients: algorithm and rounding error analysis

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

### Pipelined BiCGStab: algorithm and numerical stability

- ▶ pipe-BiCGStab for non-symmetric operators (alternative to pipe-GMRES)

## Related work

---

### Longer pipelines: pipe( $\ell$ )-GMRES & pipe( $\ell$ )-CG

talk by W. Vanroose

- ▶ useful when  $\text{time}(\text{GLRED})$  is much larger than  $\text{time}(\text{SPMV} + \text{PREC})$
- ▶ further reduced numerical stability (WIP)

### Shifted pipelined CG

- ▶ introduce shift  $\sigma$  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









### Soft error detection

with INRIA Bordeaux

- ▶ rounding error model allows separation rounding errors vs. soft errors
- ▶ detect-and-correct strategy improves soft error resilience



## Conclusion 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( $\ell$ )-CG** and **pipe( $\ell$ )-GMRES** in PETSc is work in progress.