Pipelined Sparse Solvers: Can More Reliable Computations Help Us to Converge Faster?

Roman Iakymchuk
joint work with
José Aliaga (UJI), Stef Graillat (SU), Mykhailo Havdiak (LNU)

Uppsala University and Umeå University
Sweden
roman.iakymchuk@it.uu.se

SIAM PP 2024
Baltimore, MD, USA
March 5-8, 2024

Motivation

Accuracy and Reproducibility of Preconditioned Conjugate Gradient

Matrix	$cond(A)$	MPI@MN4	MPI+OMP@MN4	MPI	MPI+OMP
`gyro_k`	$1.10e+09$	16,557	16,064	16,518	16,623

Iterations converge with $tol = 10^{-8}$ for the `gyro_k` matrix from SuiteSparse

Motivation

Accuracy and Reproducibility of Preconditioned Conjugate Gradient

$$\boxed{Ax = b}$$

while $(\tau > \tau_{\max})$

Step	Operation
$S1:$	$w := Ad$
$S2:$	$\rho := \beta / < d, w >$
$S3:$	$x := x + \rho d$
$S4:$	$r := r - \rho w$
$S5:$	$z := M^{-1} r$
$S6:$	$\beta := < z, r >$
$S7:$	$d := (\beta / \beta_{old}) d + z$
$S8:$	$\tau := < r, r >$

end while

CG Residual – Step S8 – $\sqrt{\sum_{i=0}^{N-1} r_i^2}$

Iter	Sequential	Parallel w 48 cores
0	0x1.19f179eb7f033p+49	0x1.19f179eb7f033p+49
2	0x1.f86089ece**5bd4**p+38	0x1.f86089ece**af76**p+38
9	0x1.fc59a29d3**599a**p+28	0x1.fc59a29d3**2d1b**p+28
10	0x1.74f5ccc**1d03cb**p+22	0x1.74f5ccc**201246**p+22
...	...	...
40	0x1.7031058**dd6bcf**p-19	0x1.7031058**eaf4c2**p-19
42	0x1.4828f76**d1aa3**p-23	0x1.4828f76**bda71a**p-23
45	0x1.8646260a**2dae8**p-26	0x1.8646260a**6da06**p-26
47	0x1.13fa97e**1e76bf**p-33	0x1.13fa97**e240f7c**p-33

The matrix is from the finite-difference method of a 3D Poisson's equation with 27 stencil points, $cond(A) = 10^{12}$, n=4,019,679, $tol = 10^{-8}$.

Outline

Computer arithmetic & floating-point numbers

Reproducibility/ Robustness of LA kernels and solvers

Pipelined BiCGStab with residual replacement

Summary

Outline

Floating-point arithmetic

- Most real numbers cannot be stored exactly; they need to be *rounded* and *bounded* (**round-off errors**)
- Almost all computer hardware and software support the **IEEE Standard for Floating-Point Arithmetic** IEEE 754
- IEEE 754 adopted in 1985: formats and operations $(+, -, *, /)$
  - Before 1985: each vendor had its own base and formats
  - Revised in 2008: $fma(a, b, c) = a * b + c$ with one rounding
  - Latest version IEEE 754-2019 includes binary16
- Yields a machine-independent model of how floating-point arithmetic behaves

Non-associativity

- Floating-point operations $(+, \times)$ are commutative but non-associative
  - $(a + b) + c \neq a + (b + c)$
  - $(-1 + 1) + 2^{-53} \neq -1 + (1 + 2^{-53})$ in double precision

$$1 = (1.\underbrace{0000000000000000000000000000000000000000000000000000}_{1\ldots52})_2$$

$$2^{-53} = (0.\underbrace{0000000000000000000000000000000000000000000000000001}_{1\ldots52})_2$$

Non-associativity

- Floating-point operations $(+, \times)$ are commutative but non-associative
  - $(a + b) + c \neq a + (b + c)$
  - $(-1 + 1) + 2^{-53} \neq -1 + (1 + 2^{-53})$ in double precision

$$1 = (1.\underbrace{0000000000000000000000000000000000000000000000000000}_{1\ldots52})_2$$

$$2^{-53} = (0.\underbrace{0000000000000000000000000000000000000000000000000001}_{1\ldots52})_2$$

- Another example is summation in ascending or descending orders
- Consequence: results of floating-point computations depend on the order of computation especially in parallel

Outline

Accurate and reproducible computing

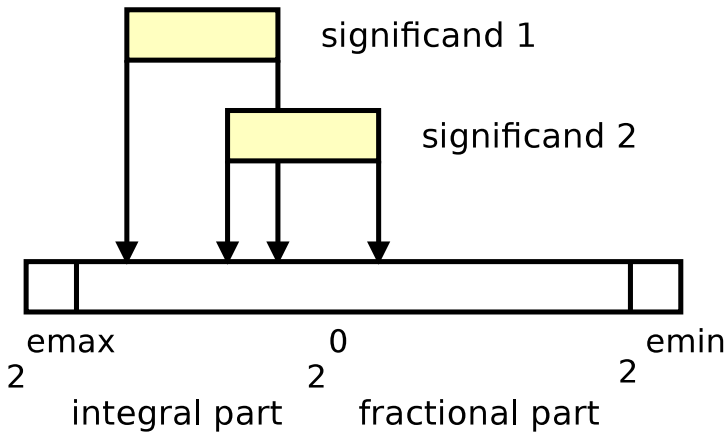
- **Reproducibility** – ability to obtain bit-wise identical and accurate results from run-to-run on the same input data on the same or different architectures

Challenges

- More heterogenous parallelism of current computers
  → GPU accelerators, etc.
- A high number of floating-point operations performed
  → Each of them leads to a round-off error
- Lack of deterministic execution
  → Dynamic scheduling and compiler optimisation
  → Different rounding modes, e.g. in TensorCore

Control of errors (1/2)

- "Infinite" precision: reproducible independently from the inputs
- Example: Kulisch accumulator



- Large register of size 2,097 (1,022 + 1,023 + 52) bits
- Divided into digits of size 64 bits stored as unsigned integers
- Due to its size and indirect memory access, Kulisch accumulator is expensive

Control of errors (2/2)

- Error-Free Transformations (EFT) for summation

Algorithm 1 (Møller-Knuth-Dekker)

$(r, s) = \texttt{2sum}(a, b)$

1: $r \leftarrow a + b$
2: $z \leftarrow r - a$
3: $s \leftarrow (a - (r - z)) + (b - z)$

Algorithm 2 ($|a| \geq |b|$)

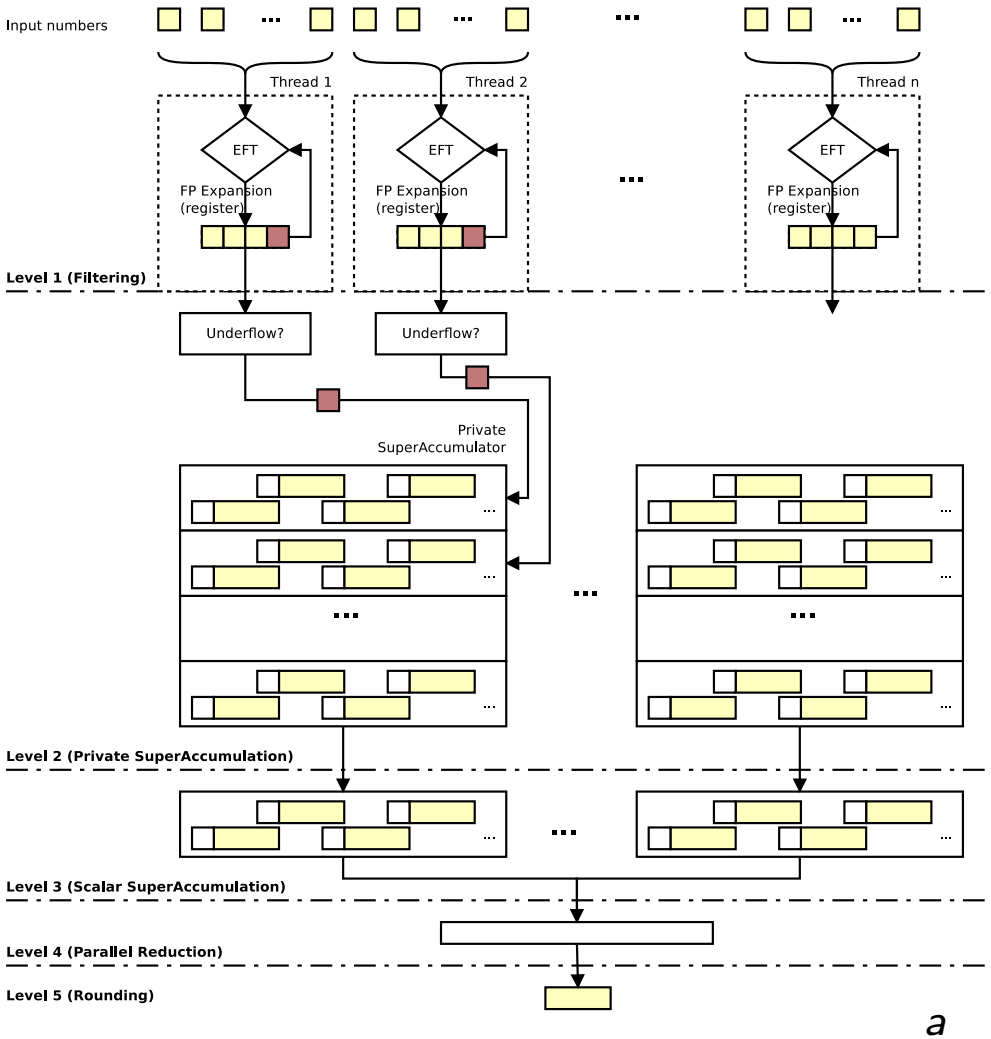
$(r, s) = \texttt{fast2sum}(a, b)$

1: $r \leftarrow a + b$
2: $z \leftarrow r - a$
3: $s \leftarrow b - z$

- EFT gives access to the rounding errors of individual operations
- Store the result and the error in a short array of the same type as parameters – FP Expansions (FPE)

→ Example: double-double or quad-double
(work well on a set of relatively close numbers)

Reproducible parallel reduction



a

- Based on FPE with EFT and Kulisch accumulator
- Suitable for CPUs, GPUs, Xeon Phi
- Guarantees "inf" precision

→ **bit-wise reproducibility**

[a] S. Collange et al. Numerical Reproducibility for the Parallel Reduction on Multi- and Many-Core Architectures. ParCo, 49, 2015, 83-97

Preconditioned BiCGStab

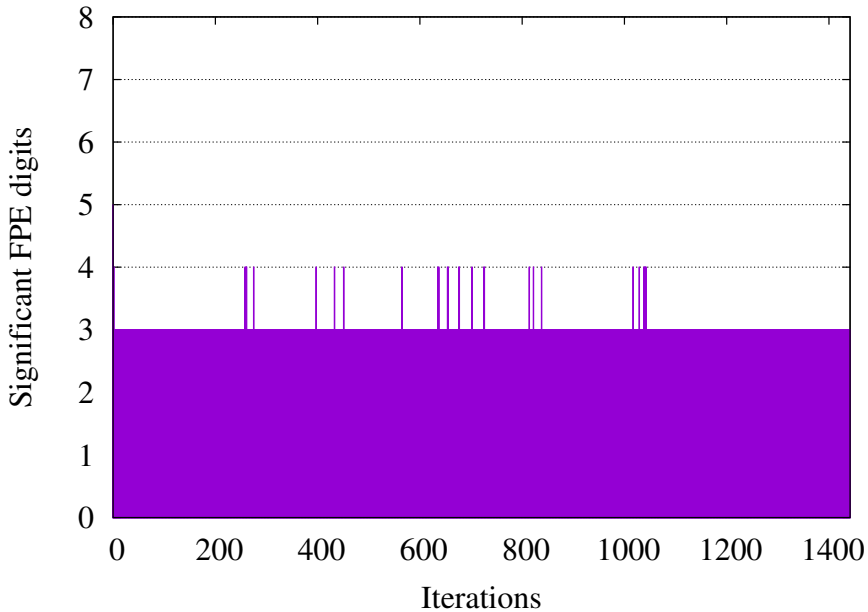
$$\boxed{Ax = b}$$

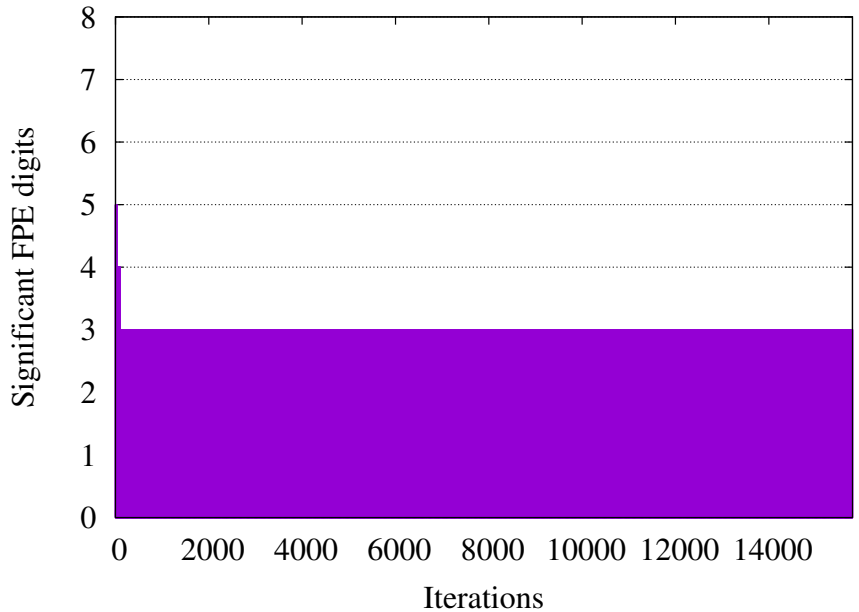
while $(\tau > \tau_{\max})$

Step	Operation	Kernel	Comm
$S1:$	$\hat{p}^j := M^{-1}p^j$	Apply precond.	–
$S2:$	$s^j := A\hat{p}^j$	spmv	Alltoallw
$S3:$	$\alpha^j := \langle r^0, r^j\rangle / \langle r^0, s^j\rangle$	dot product	Allreduce
$S4:$	$q^j := r^j - \alpha^j s^j$	axpy-like	–
$S5:$	$\hat{q}^j := M^{-1}q^j$	Apply precond.	–
$S6:$	$y^j := A\hat{q}^j$	spmv	Alltoallw
$S7:$	$\omega^j := \langle q^j, y^j\rangle / \langle y^j, y^j\rangle$	Two dot products	Allreduce
$S8:$	$x^{j+1} := x^j + \alpha^j\hat{p}^j + \omega^j\hat{q}^j$	Two axpy	–
$S9:$	$r^{j+1} := q^j - \omega^j y^j$	axpy-like	–
$S10:$	$\beta^j := \frac{\langle r^0, r^{j+1}\rangle}{\langle r^0, r^j\rangle} * \frac{\alpha^j}{\omega^j}$	dot product	Allreduce
$S11:$	$\tau^{j+1} := \parallel r^{j+1} \parallel_2$	dot product ($l2$ norm)	Allreduce
$S12:$	$p^{j+1} := r^{j+1} + \beta^j(p^j - \omega^j s^j)$	Two axpy-like	–

end while

Reproducibility: required precision





`msc01050` (Boeing)
$NNZ = 26,198$
$cond(A) = 9.0e+15$

`gyro_k` (Oberwolfach)
$NNZ = 1,021,159$
$cond(A) = 1.1e+09$

"*Reproducibility of Parallel Preconditioned Conjugate Gradient in Hybrid Parallel Environments*" Roman Iakymchuk, Maria Barreda, Stef Graillat, José I Aliaga, Enrique S Quintana-Orti. IJHPCA, FirstOnline June 17 2020, vol 34, issue 5, pp. 502-518.

Re-assuring Reproducibility in Sparse Solvers

Sources of non-reproducibility

- parallel reduction: `dot product` with `MPI_Allreduce`
- compiler auto-replacement of $ax + b$ in favor of fma (`axpy`)
- $a * b + c * d * e$ with or without fma (`spmv`)

Re-assuring Reproducibility in Sparse Solvers

Sources of non-reproducibility

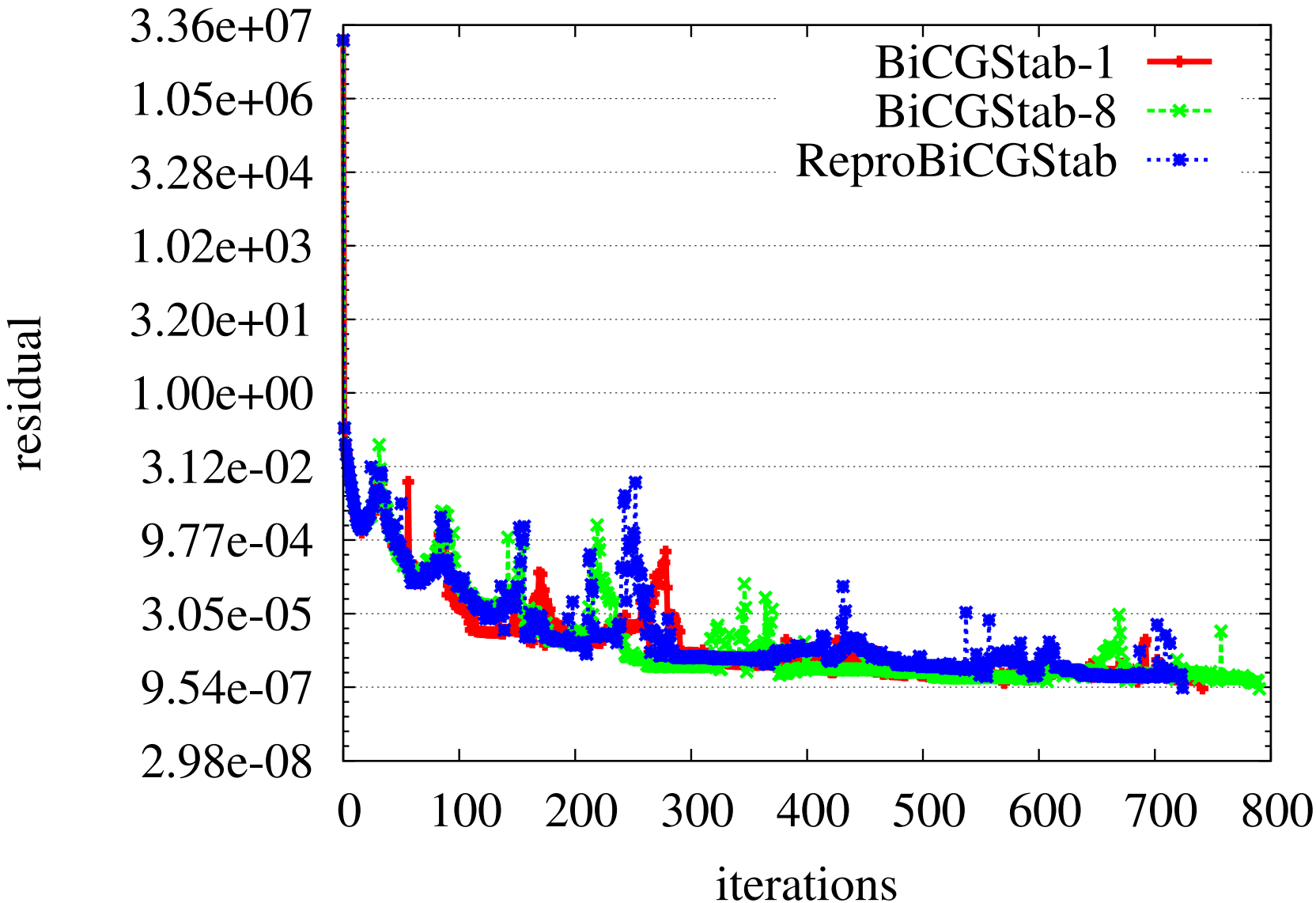
- parallel reduction: `dot product` with `MPI_Allreduce`
- compiler auto-replacement of $ax + b$ in favor of fma (`axpy`)
- $a * b + c * d * e$ with or without fma (`spmv`)

Solutions

- **Combine arithmetic solutions, reorganization of operations, and sequential executions**
  - $\rightarrow$ aiming for lighter or lightweight approaches
- `spmv` computes blocks of rows in parallel, but with $a * b + / - c * d$
  - $\rightarrow$ ensure deterministic execution with explicit `fma`
- `axpy` relies explicitly on `fma`
- accurate and reproducible `dot`
  - $\rightarrow$ ExBLAS-based approach
  - $\rightarrow$ FPE with size 8 and early-exit
- $b = Ad$ $d = \frac{1}{\sqrt{N}}(1, \ldots, 1)^T \rightarrow b = Ad$ and $b = \frac{1}{\sqrt{N}}b$

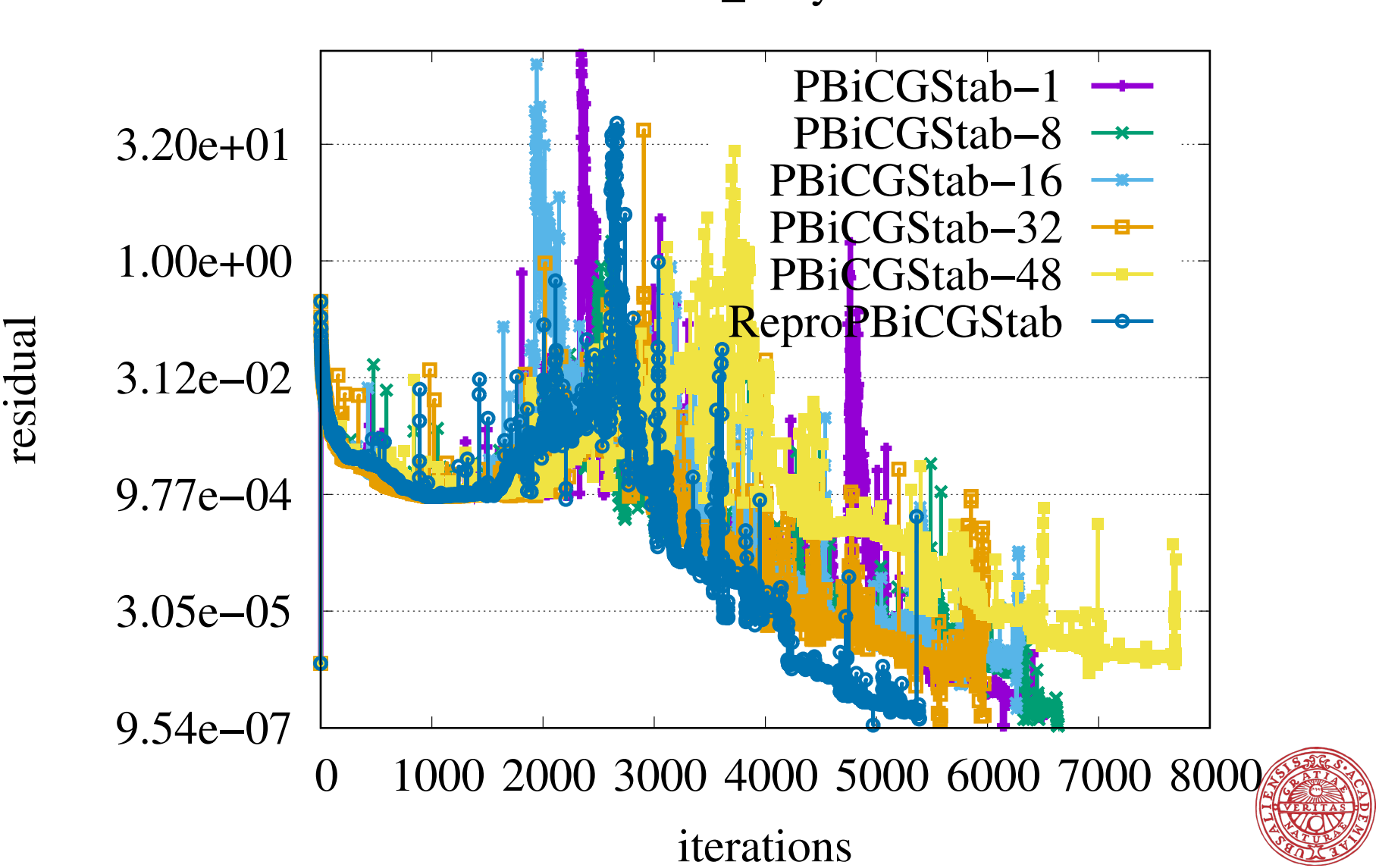
BiCGStab: convergence

Residual history for *fs_760_3d* (5.8K nnz, w/o precond) $tol = 10^{-6}$



PBiCGStab: convergence

Residual history for *tmt_unsym* (4.58M nnz) $tol = 10^{-6}$



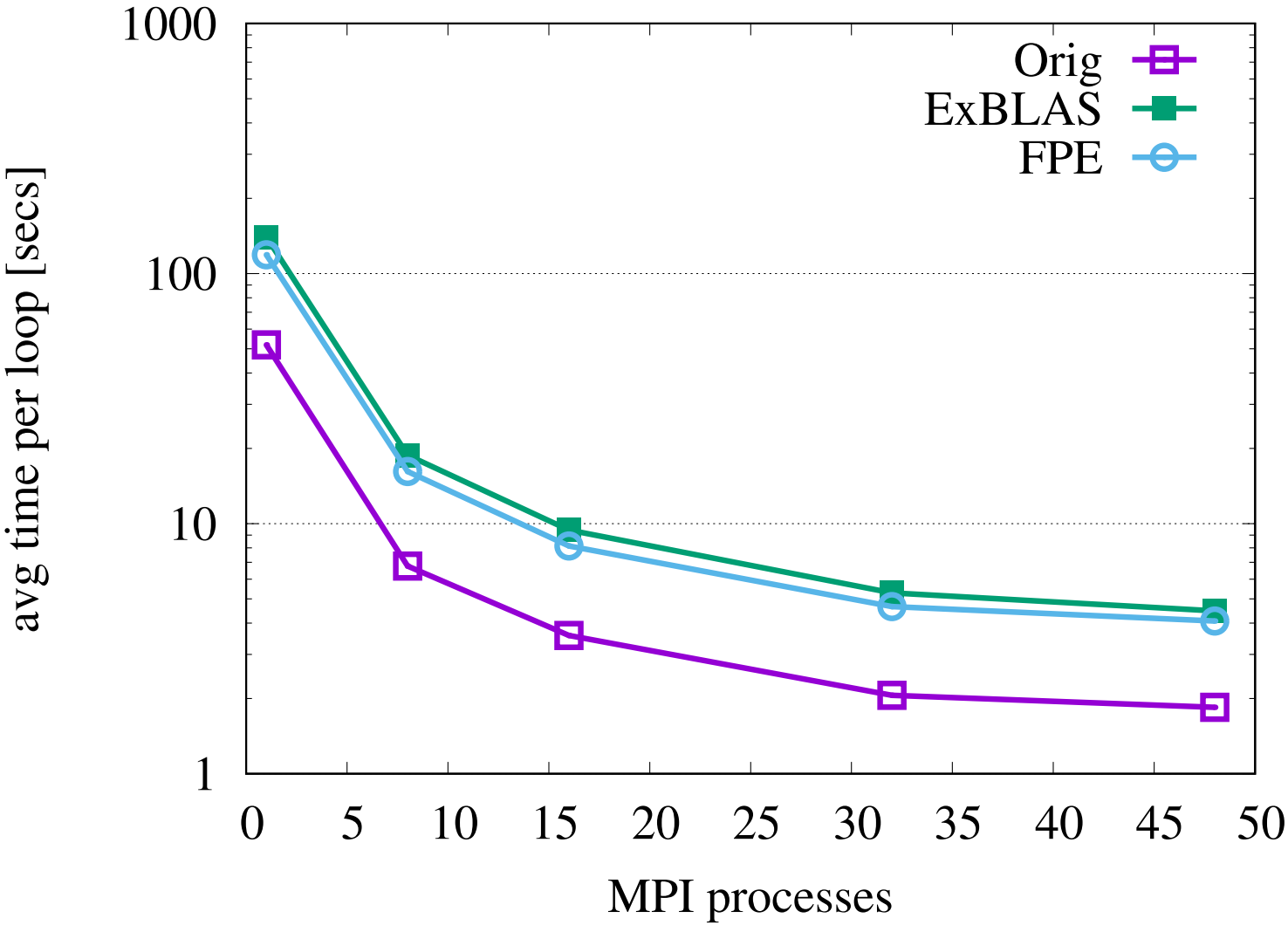
PBiCGStab: convergence & reproducibility

Iteration	Residual			
	MPFR	*Original* 1 proc	*Original* 8 procs	*Exblas & FPE*
0	0x1.3566ea57eaf3fp+2	0x1.3566ea57ea**b49**p+2	0x1.3566ea57ea**b49**p+2	0x1.3566ea57eaf3fp+2
1	0x1.146d37f18fbd9p+0	0x1.146d37f18f**aaf**p+0	0x1.146d37f18f**ab**p+0	0x1.146d37f18fbd9p+0
...	...	...	...	...
99	0x1.cedf0ff322158p-13	**0x1.88008701ba87p-12**	**0x1.04e23203fa6fcp-12**	0x1.cedf0ff322158p-13
100	0x1.be3698f1968cdp-13	**0x1.55418acf1af27p-12**	**0x1.fbf5d3a5d1e49p-13**	0x1.be3698f1968cdp-13
...	...	...	...	...
208	0x1.355b0f18f5ac1p-20	**0x1.19edf2c932ab8p-18**	**0x1.b051edae310c7p-20**	0x1.355b0f18f5ac1p-20
209	0x1.114dc7c9b6d38p-20	**0x1.19b74e383f74ep-18**	**0x1.a18fc929018d4p-20**	0x1.114dc7c9b6d38p-20
210	0x1.03b1920a49a7ap-20	**0x1.19c846848f361p-18**	**0x1.c7eb5bbc198b1p-20**	0x1.03b1920a49a7ap-20

Table: Accuracy and reproducibility of the intermediate and final residual against MPFR for the *orsreg_1* matrix ($cond(A) = 6.7e+03$, 14K nnz).

PBiCGStab: performance (1/2)

Strong scaling for *Queen_4147* (316.5M nnz) $tol = 10^{-6}$

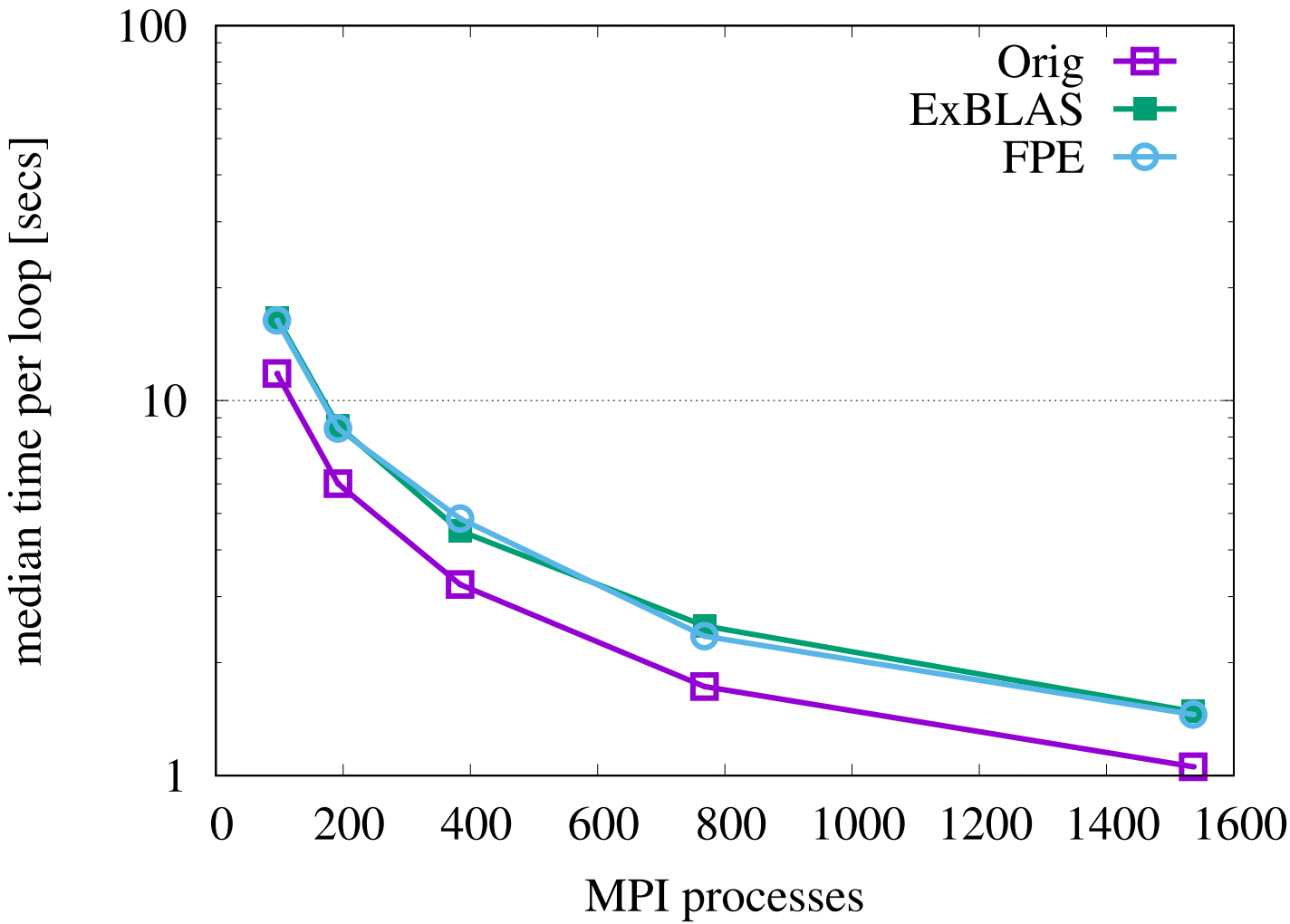


2x24 core Intel Xeon Gold 6240R CPU @2.4 GHz

PBiCGStab: performance (2/2)

3D Poisson's equation with 27 stencil points: $n = 16M$ and $tol = 10^{-8}$

perturbed matrix $(1 - 10^{-4})$ aka the e-type model



Strong scaling on 32 Intel Xeon Gold 6240R nodes

Outline

Preconditioned pipelined BiCGStab

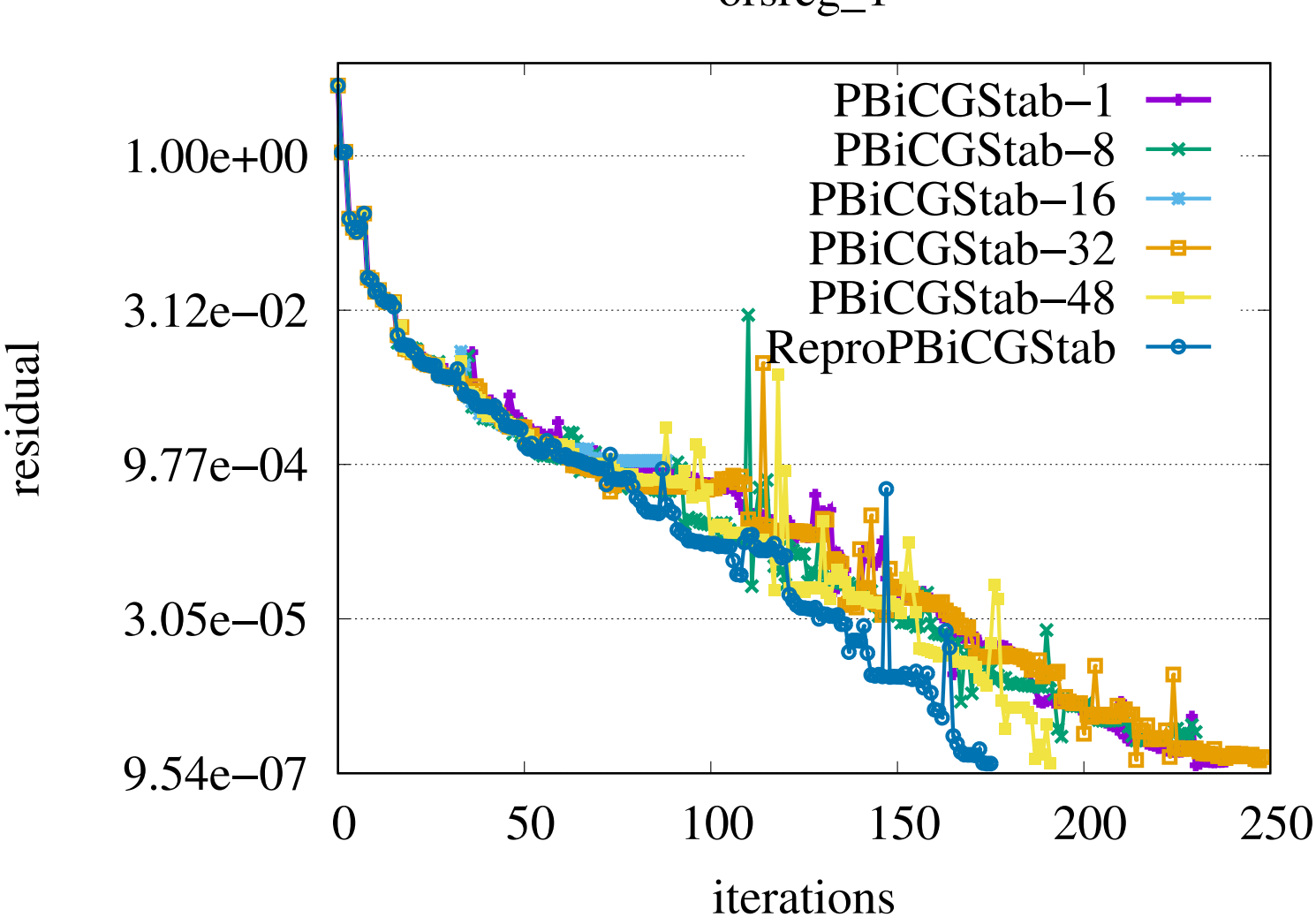
while $(\tau > \tau_{\max})$

Step	Operation	Kernel	Comm
$S1:$	$\hat{p}^j := \hat{r}^j + \beta^{j-1}(\hat{p}^{j-1} - \omega^{j-1}\hat{s}^{j-1})$	Two **axpy**-like	–
$S2:$	$s^j := w^j + \beta^{j-1}(s^{j-1} - \omega^{j-1}z^{j-1})$	Two **axpy**-like	–
$S3:$	$\hat{s}^j := \hat{w}^j + \beta^{j-1}(\hat{s}^{j-1} - \omega^{j-1}\hat{z}^{j-1})$	Two **axpy**-like	–
$S4:$	$z^j := t^j + \beta^{j-1}(z^{j-1} - \omega^{j-1}v^{j-1})$	Two **axpy**-like	–
$S5:$	$q^j := r^j - \alpha^j s^j$	**axpy**-like	–
$S6:$	$\hat{q}^j := \hat{r}^j - \alpha^j \hat{s}^j$	**axpy**-like	–
$S7:$	$y^j := w^j - \alpha^j z^j$	**axpy**-like	–
$S8:$	$\langle q^j, y^j\rangle, \langle y^j, y^j\rangle$	Two **dot** products	Iallreduce
$S9:$	$\hat{z}^j := M^{-1}z^j$	Apply precond.	–
$S10:$	$v^j := A\hat{z}^j$	**spmv**	Alltoallw
$S11:$	$\omega^j := \langle q^j, y^j\rangle / \langle y^j, y^j\rangle$	Two **dot** products	Wait for S8
$S12:$	$x^{j+1} := x^j + \alpha^j\hat{p}^j + \omega^j\hat{q}^j$	Two **axpy**	–
$S13:$	$r^{j+1} := q^j - \omega^j y^j$	**axpy**-like	–
$S14:$	$\hat{r}^{j+1} := \hat{q}^j - \omega^j(\hat{w}^j - \alpha^j\hat{z}^j)$	Two **axpy**-like	–
$S15:$	$w^{j+1} := y^j - \omega^j(t^j - \alpha^j v^j)$	Two **axpy**-like	–
$S16:$	$\langle r^0, r^{j+1}\rangle, \langle r^0, w^{j+1}\rangle$ $\langle r^0, s^j\rangle, \langle r^0, z^j\rangle$	Four **dot** products	Iallreduce
$S17:$	$\hat{w}^{j+1} := M^{-1}w^{j+1}$	Apply precond.	–
$S18:$	$t^{j+1} := A\hat{w}^{j+1}$	**spmv**	Alltoallw
$S19:$	$\beta^j := \frac{\langle r^0, r^{j+1}\rangle}{\langle r^0, r^j\rangle} * \frac{\alpha^j}{\omega^j}$ $\alpha^{j+1} := \frac{\langle r^0, r^{j+1}\rangle}{\langle r^0, w^{j+1}\rangle + \beta^j\langle r^0, s^j\rangle - \beta^j\omega^j\langle r^0, z^j\rangle}$	Four **dot** products	Wait for S16

end while

p-BiCGStab: convergence

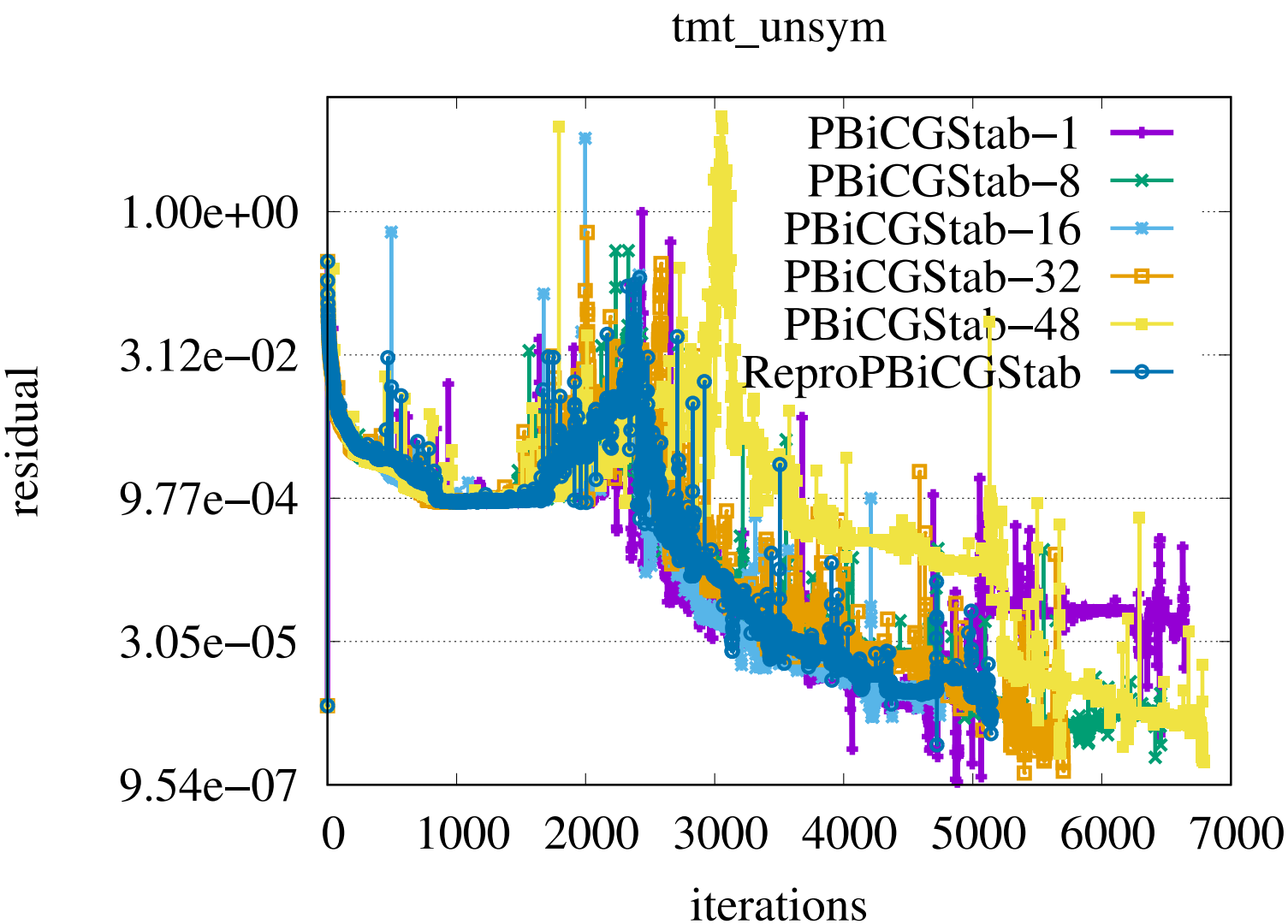
Residual history for *orsreg_1* (2.2K nnz) $tol = 10^{-6}$



2x24 core Intel Xeon Gold 6240R CPU @2.4 GHz

p-BiCGStab: convergence

Residual history for *tmt_unsym* (918K nnz) $tol = 10^{-6}$

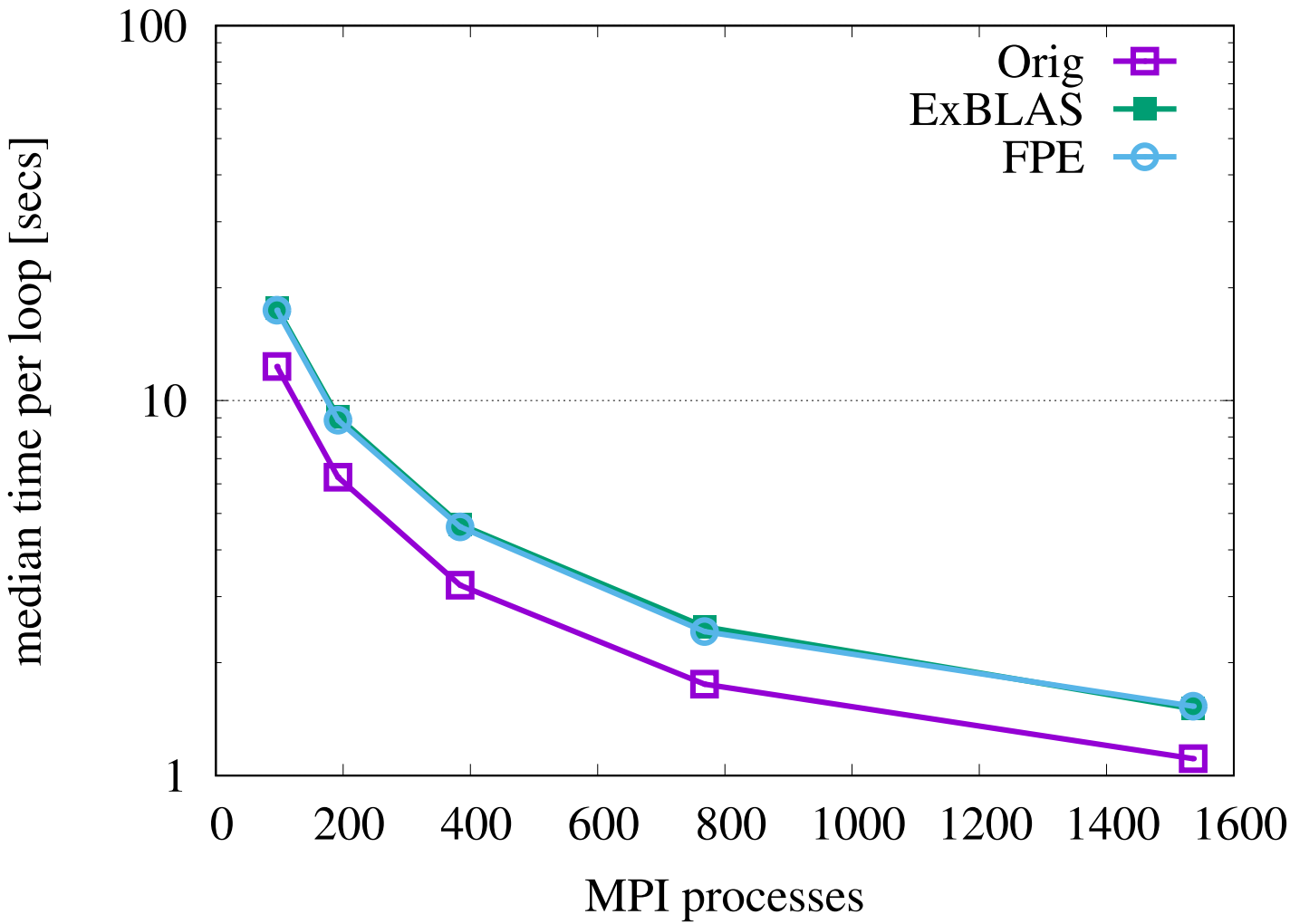


2x24 core Intel Xeon Gold 6240R CPU @2.4 GHz

p-BiCGStab: performance

3D Poisson's equation with 27 stencil points: $n = 64M$ and $tol = 10^{-8}$

perturbed matrix $(1 - 10^{-4})$ aka the e-type model



Strong scaling on 32 Intel Xeon Gold 6240R nodes

p-BiCGStab: residual replacement

- Max attainable accuracy below $\|r_i\|/\|r_0\| \leq 10^{-6}$ can be an issue for p-BiCGStab
- *"Several orders of magnitude on maximal attainable precision are typically lost when switching to the pipelined algorithm."*[a]
- Cause: 11 axpy operations $\rightarrow$ amplifying local rounding errors

p-BiCGStab: residual replacement

- Max attainable accuracy below $\|r_i\|/\|r_0\| \leq 10^{-6}$ can be an issue for p-BiCGStab
- *"Several orders of magnitude on maximal attainable precision are typically lost when switching to the pipelined algorithm."*[a]
- Cause: 11 axpy operations $\rightarrow$ amplifying local rounding errors
- Remedy 1: residual replacement strategy to reset residual r_i and $\hat{r}_i$ to their true values every k iterations

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

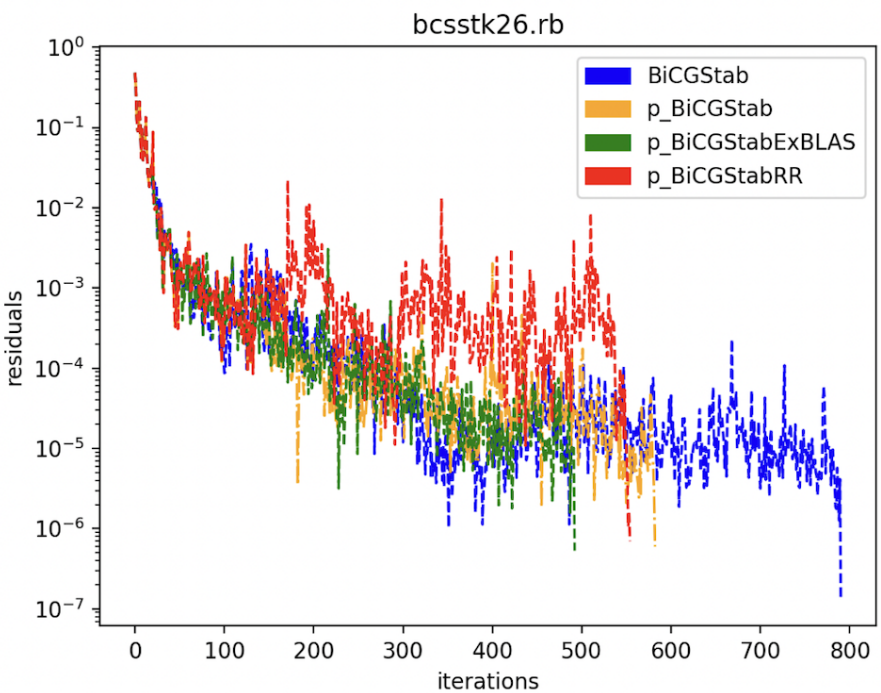
- Remedy 2: apply the ExBLAS approach

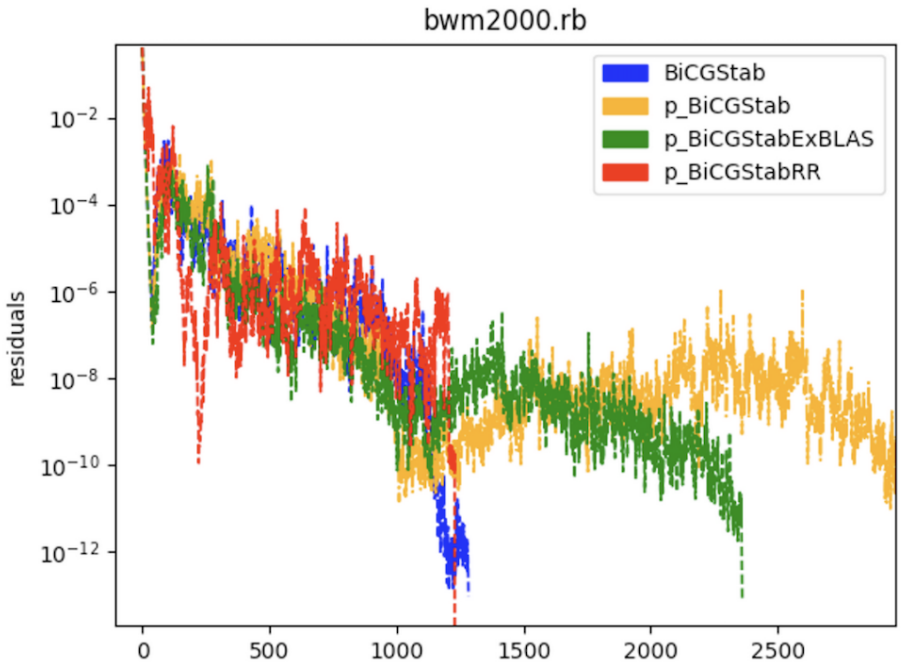
[a] S. Cools and W. Vanroose. "The communication-hiding pipelined BiCGstab method for the parallel solution of large unsymmetric linear systems". In: Parallel Computing 65 (2017), pp. 1–20.

p-BiCGStab w residual replacement (1/2)

convergence

Residual history for $tol = 10^{-6}$ and $tol = 10^{-13}$



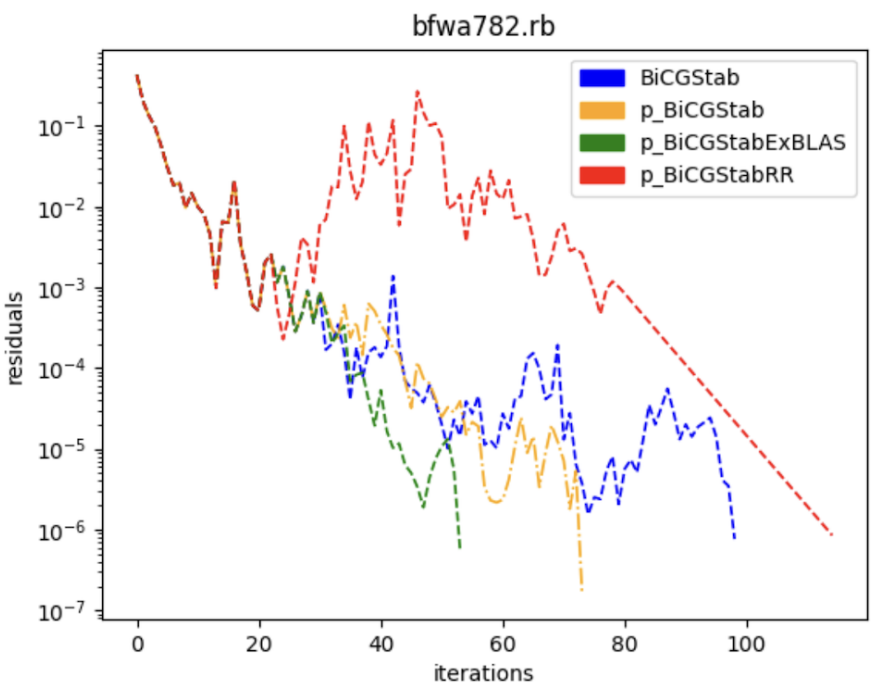


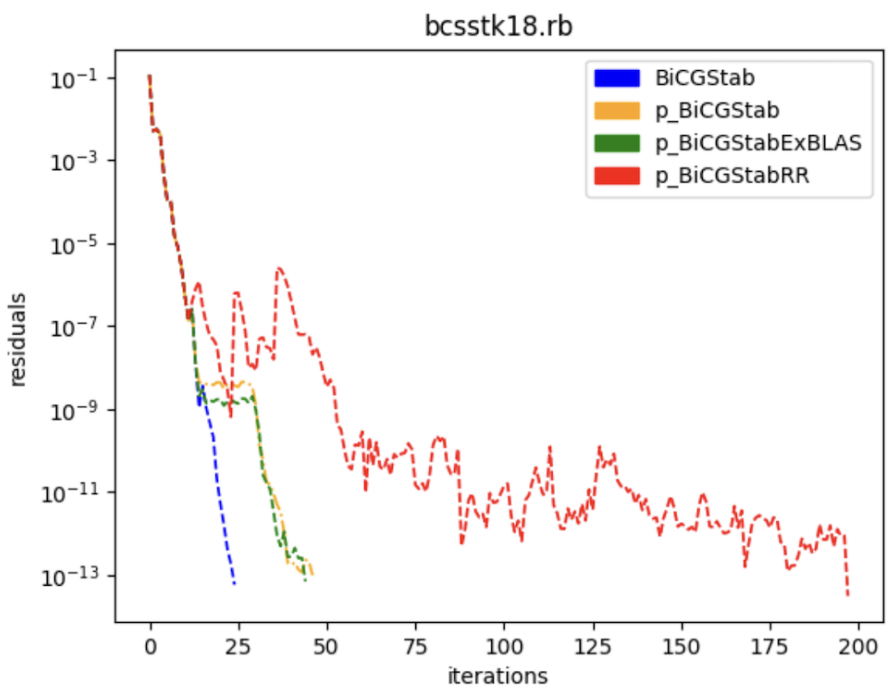
2x14 core Intel Xeon Gold 6132 CPU @2.6 GHz

p-BiCGStab w residual replacement (2/2)

convergence

Residual history for $tol = 10^{-6}$ and $tol = 10^{-13}$





2x14 core Intel Xeon Gold 6132 CPU @2.6 GHz

Outline

Verification of numerical results

How do we verify parallel scientific programs ?

Verification of numerical results

How do we verify parallel scientific programs ?

Comparison against

freq Sequential version

rare MPFR version (sequential)

rare MATLAB version

Verification of numerical results

How do we verify parallel scientific programs ?

Comparison against

freq Sequential version

rare MPFR version (sequential)

rare MATLAB version

Approaches and initiatives

- Numerical verification
- **Reproducible solvers with ExBLAS**
- Computer arithmetic tools like VerifiCarlo as part of CI/CD
  - Detect numerical abnormalities like cancellations, NaNs, etc

Summary

- Computer arithmetic operates with finite precisions
- Implement your algorithms with caution:
  - select suitable and stable algorithm
  - trade-off between more accurate or fast version
  - reduce impact of compiler optimization
- accurate and reliable computing can be reinstalled
- Future Work
  - pipelined CG and automatic choice of $k = 20 - 100$ for residual replacement
  - theoretical analysis
  - more examples

[1]This research is partially funded by EuroHPC JU CoE CEEC (No. 101093393) and IT Dept at Uppsala University

Summary

- Computer arithmetic operates with finite precisions
- Implement your algorithms with caution:
  - select suitable and stable algorithm
  - trade-off between more accurate or fast version
  - reduce impact of compiler optimization
- accurate and reliable computing can be reinstalled
- Future Work
  - pipelined CG and automatic choice of $k = 20 - 100$ for residual replacement
  - theoretical analysis
  - more examples

Thank you for your attention ![1]

[1]This research is partially funded by EuroHPC JU CoE CEEC (No. 101093393) and IT Dept at Uppsala University