Remedies for Reproducibility Issue in Conjugate Gradient Solvers

Daichi Mukunoki, Roman Iakymchuk, Fabienne Jézéquel, Katsuhisa Ozaki, Takeshi Ogita, Toshiyuki Imamura

Remedies for Reproducibility Issue in Conjugate Gradient Solvers

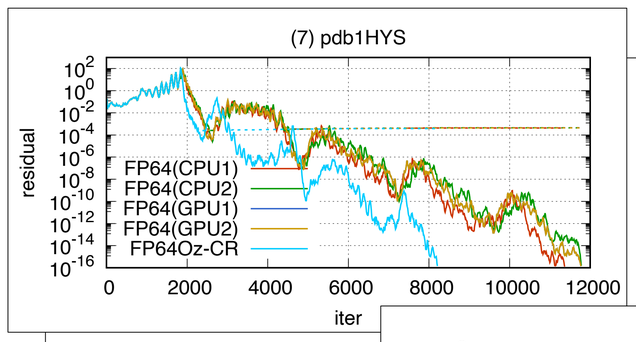
Daichi Mukunoki[1](daichi.mukunoki@riken.jp), Roman Iakymchuk[2], Fabienne Jézéquel[2,3], Katsuhisa Ozaki[4], Takeshi Ogita[5], Toshiyuki Imamura[1]
1. RIKEN Center for Computational Science, Japan, 2. Sorbonne Université, CNRS, LIP6, France, 3. Université Paris-Panthéon-Assas, France,
4. Shibaura Institute of Technology, Japan, 5. Tokyo Woman's Christian University, Japan

Introduction

- **Reproducibility issues in CG on many-cores**
  - Floating-point operations: non-associative; result may change depending on the order of computations
  - Factors to disturb reproducibility on many-cores: different number of threads, cores, SIMD width, etc., and different precision and rounding (e.g., FMA, x87)
  - In CG solvers, the parallel computation of DOT, NRM2 and SpMV and the use of FMA in AXPY are factors to disturb reproducibility
- **Needs for reproducibility**
  - Quality assurance of science & engineering; reproduction of result on different environments and by third party
  - Debugging and porting
- **Our poster**
  - Exploring remedies for reproducibility issues in CG solvers on many-core architectures
  - Introducing three different approaches developed by us

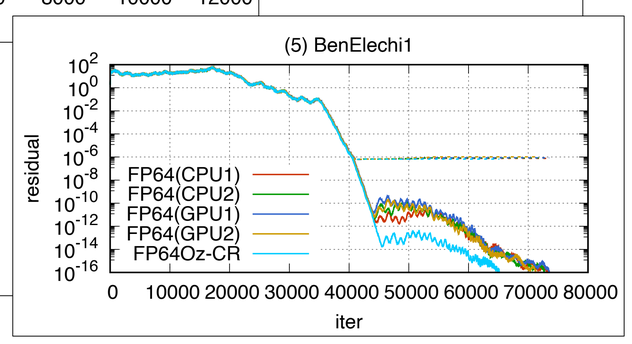
Algorithm CG method solving $\boldsymbol{Ax} = \boldsymbol{b}$ (subscript 'i' means number of iterations)

1: $\boldsymbol{p}_0 = \boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$ // SpMV
2: $\rho_0 = {\boldsymbol{r}_0}^T \boldsymbol{r}_0$ // DOT
3: $i = 0$
4: **while** 1 **do**
5: $\quad \boldsymbol{q}_i = \boldsymbol{A}\boldsymbol{p}_i$ // SpMV
6: $\quad \alpha_i = \rho_i / {\boldsymbol{p}_i}^T \boldsymbol{q}_i$ // DOT
7: $\quad \boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \alpha_i \boldsymbol{p}_i$ // AXPY
8: $\quad \boldsymbol{r}_{i+1} = \boldsymbol{r}_i - \alpha_i \boldsymbol{q}_i$ // AXPY
9: $\quad$ **if** $||\boldsymbol{r}_{i+1}||/||\boldsymbol{b}|| < \epsilon$ **then** // NRM2
10: $\quad\quad$ **break**
11: $\quad$ **end if**
12: $\quad \rho_{i+1} = {\boldsymbol{r}_{i+1}}^T \boldsymbol{r}_{i+1}$ // DOT
13: $\quad \beta_i = \rho_{i+1} / \rho_i$
14: $\quad \boldsymbol{p}_{i+1} = \boldsymbol{r}_{i+1} + \beta_i \boldsymbol{p}_i$ // SCAL, AXPY
15: $\quad i = i + 1$
16: **end while**



- CPU1: XeonGold6126
- CPU2: XeonPhi7250
- GPU1: Tesla V100
- GPU2: Tesla P100
- FP64Oz-CR: reproducible CG based on OzBLAS[1]

- $\boldsymbol{b}$=$\boldsymbol{x}_0$=(1, …, 1)^T
- Stopping criterion: 1e-16
- Relative residual $||\boldsymbol{r}_i||/||\boldsymbol{b}||$ (solid line)
- Relative true residual $||\boldsymbol{b}\text{-}\boldsymbol{A}\boldsymbol{x}_i||/||\boldsymbol{b}||$ (dotted line)



Remedies

Algorithm of unpreconditioned CG
DOT, NRM2 & SpMV may disturb reproducibility in parallel computation

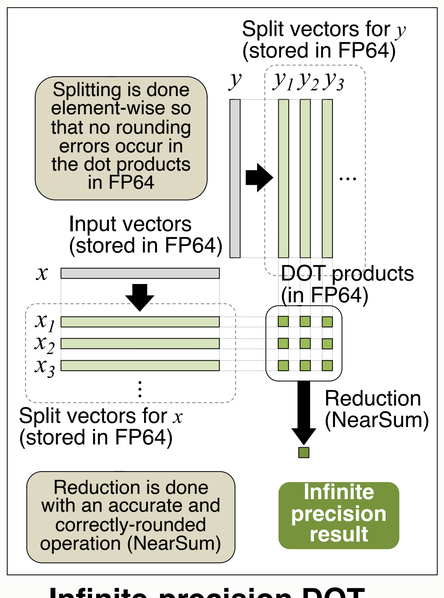
Examples of reproducibility issue in CG [1]
(on matrices from SuiteSparse [2])

Method	Effect for reproducibility	Project	Remarks
Infinite-precision BLAS	**Bit-wise reproducibility is 100% ensured**	**OzBLAS** [3], **ExBLAS** [8], RARE-BLAS [7]	Solution for BLAS operations only at present
Reproducible BLAS (non infinite-precision)		ReproBLAS [6], OzBLAS*	
Fixed-order computation		Conditional Numerical Reproducibility (CNR) [15] in MKL	CNR is a solution for BLAS operations only and under limited conditions (some limited CPUs with MKL)
Accurate computation (non infinite-precision)	Possibility of reproducibility as well as number of reproducible digits are increased	Some high-precision arithmetic libraries (e.g., MPFR [9]) and BLAS (e.g., MPLAPACK [10])	Can be applied by code conversion (even to preconditioners)
Numerical validation (with stochastic arithmetic)	The digits unaffected by rounding errors (which are considered reproducible) are evaluated probabilistically	**CADNA** [11], Verrou [12], Verificarlo [13], etc.	

Reproducible CG solver with infinite-precision BLAS (OzBLAS)

Bit-wise reproducibility is 100% ensured with infinite-precision DOT & SpMV

- OzBLAS [3] ensures reproducibility through infinite-precision inner product using Ozaki scheme [4] with correctly-rounded summation NearSum [5] (tunable accuracy is also possible).
- On FP64 data, OzBLAS-based solvers [1] take 2.5-17x overheads v.s. standard solvers (using MKL on CPUs / cuBLAS+cuSparse on GPUs), but the overhead is problem dependent



Infinite-precision DOT using Ozaki scheme [4]

Std. CG vs Repro. CG with OzBLAS (FP64Oz)
- CPU: Xeon Platinum 8360Y (Icelake, 1.4TF, 205GB/s)
- GPU: A100-SXM4-40GB (Ampere, 19.5TF, 1.6TB/s)
- b=x_0=(1,…,1)^T, stopping criterion: 1e-16

Matrices [2]	Number of iterations			Overhead of Repro to Std (in times)	
	Std		Repro		
	(CPU)	(GPU)	(both)	(CPU)	(GPU)
tmt_sym	7839	7822	7793	17.0	11.6
gridgena	2400	2468	2393	5.1	5.7
cfd1	3276	3278	3279	4.2	5.4
cbuckle	23834	23950	23724	9.0	5.3
BenElechi1	72972	71521	65161	6.3	5.2
gyro_k	60545	61238	46641	6.6	4.0
pdb1HYS	10998	11841	8214	5.3	3.1
nd24k	15458	14689	12837	2.5	3.3

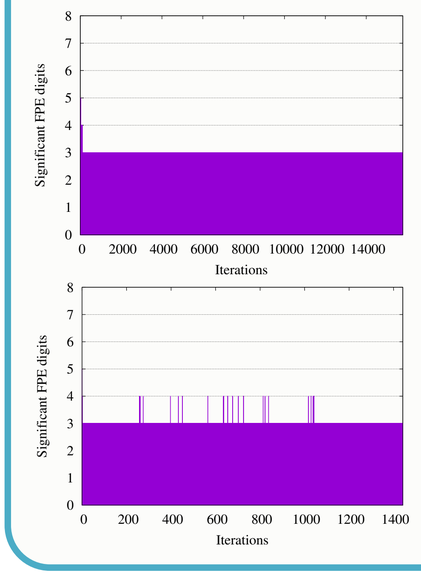
OzBLAS: https://www.r-ccs.riken.jp/labs/lpnctrt/projects/ozblas/

Accurate CG solver with ExBLAS-FPE8

Possibility of reproducibility and number of reproducible digits are increased

- ExBLAS [8] has two variants: (1) infinite-precision mode with Kulisch accumulator plus floating-point expansion (FPE) and (2) accurate mode with FPE only.
- On CG with Jacobi preconditioner, using eight FPEs (FPE8 = 8*53 bits) with the early-exit technique (avoid propagating zeros) achieves the same residual and the number of iterations in different environments. This variant is similar to the K-fold Dot (DotK [18] with K=8).

Required precision
to store every bit of information on the example of gyro_k (top) and msc01050 (bottom)



Std. PCG v.s. Acc. CG with ExBLAS-FPE8
- CPU: Xeon Platinum 8160 on MareNostrum4
- b=x_0=(1,…,1)^T, stopping criterion: 1e-8

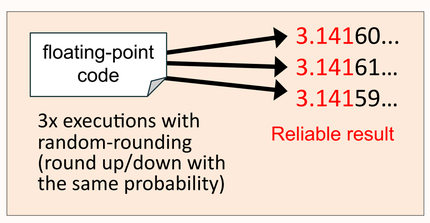
Matrices [2]	Num of iterations		Execution time (secs)		Exec time overhead v.s. Std. (times)
	Std.	FPE8	Std.	FPE8	
msc01050	1459	1441	1.2E-01	3.5E-01	2.7
olafu	44872	42342	1.4E+01	2.2E+01	1.7
bcsstk27	331	331	2.0E-02	4.5E-02	1.9
plat1919	29133	28347	2.1E+00	7.2E+00	3.5
sts4098	668	668	6.4E-02	1.8E-01	2.9
gyro_k	16623	16075	5.3E+00	8.7E+00	1.6
bcsstk28	5736	5483	8.4E-01	1.7E+00	2.0
msc04515	5138	4885	4.9E-01	1.5E+00	3.0

ExBLAS: https://github.com/riakymch/exblas

Numerical validation of CG with CADNA

The digits unaffected by rounding errors (which are considered reproducible) are evaluated probabilistically.

- Discrete Stochastic Arithmetic (DSA) [14] estimates rounding errors (i.e., number of correct digits in the result) with 95% confidence level by executing the code 3x with random-rounding; the validated result can be considered as a reproducible result
- CADNA [11] is a DSA implementation with instability detections for CPUs (Fortran/C/C++ with OpenMP/MPI) & GPUs, which validates floating-point codes with code-conversion; no development requires.
- CADNA takes large execution time overhead on many-cores; however, we are now working on new method to avoid DSA for BLAS routines [17];



Num iter	$\|\|\boldsymbol{r}_i\|\|/\|\|\boldsymbol{b}\|\|$	$\|\|\boldsymbol{b}\text{-}\boldsymbol{A}\boldsymbol{x}_i\|\|/\|\|\boldsymbol{b}\|\|$	$\|\|\boldsymbol{x}_i\|\|$ with CADNA	$\|\|\boldsymbol{r}_i\|\|$ with CADNA
1	2.70E+00	2.70E+00	0.008411791	0.62523271
2	4.89E+00	4.89E+00	0.037022818	1.13046632
3	2.92E+01	2.92E+01	0.037103746	6.75308893
:	:	:	:	:
11	1.18E+02	1.18E+02	0.076835244	27.3152207
12	8.82E+01	8.82E+01	0.077927814	20.395197
13	9.82E+01	9.82E+01	0.079094289	22.7124262
14	2.60E+01	2.60E+01	0.080217005	6.00787752
:	:	:	:	:
19	9.01E+00	9.01E+00	0.14700426	2.08438
20	4.81E+02	4.80E+02	0.2	@.0
21	1.46E+01	1.43E+01	0.27882	@.0
22	4.10E+01	3.93E+01	0.278	@.0
:	:	:	:	:
8712	3.37E-16	1.22E-06	26.9999999	@.0
8713	7.21E-17	1.23E-06	26.9999999	@.0
8714	2.41E-17	1.23E-06	26.9999999	@.0

CADNA only shows the validated digits, which can be considered reproducible

"@.0": computed-zero; no meaningful digit contains

Numerical validation of CG with CADNA
- Matrix: nos7 from SuiteSparse [2]
- FP64, $\boldsymbol{b}$=$\boldsymbol{x}_0$=(1, …, 1)^T, stopping criterion: 1e-16

CADNA: http://cadna.lip6.fr

Discussion

- **Ongoing challenges**
  - Bit-wise reproducible preconditioners (ExBLAS implements reproducible Jacobi preconditioner [16])
  - Reducing execution time overhead (it's still high, can you accept 3-10x overhead?)
  - Development cost (numerical validation is superior in this respect)
  - Comprehensive comparison of different approaches
- **Open questions**
  - Have you ever struggled with reproducibility issue?
  - How much additional run-time cost can you pay for repeatability?
  - If you are interested in reproducibility, please contact us!

References

[1] D. Mukunoki et al., Conjugate Gradient Solvers with High Accuracy and Bit-wise Reproducibility between CPU and GPU using Ozaki scheme, HPCAsia 2021.
[2] T. A. Davis and Y. Hu, The University of Florida Sparse Matrix Collection. ACM Trans. Math. Software 38, 1, 2011.
[3] D. Mukunoki et al., Reproducible BLAS Routines with Tunable Accuracy Using Ozaki Scheme for Many-core Architectures. PPAM 2019.
[4] K. Ozaki et al., Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications, Numer. Algorithms 59(1), 2012.
[5] S. M. Rump et al., Accurate Floating-Point Summation Part II: Sign, K-Fold Faithful and Rounding to Nearest. SIAM Journal on Scientific Computing 31(2), 2009.
[6] J. Demmel, P. Ahrens, H. D. Nguyen, Efficient Reproducible Floating Point Summation and BLAS. Techrep UCB/EECS-2016-121, 2016.
[7] C. Chohra et al., Reproducible, Accurately Rounded and Efficient BLAS, Euro-Par 2016.
[8] S. Collange et al., Numerical Reproducibility for the Parallel Reduction on Multi- and Many-Core Architectures. Parallel Computing 49, 2015.
[9] L. Fousse et al., MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding, ACM TOMS 33(2), 2007.
[10] M. Nakata, "The MPACK (MBLAS/MLAPACK); a multiple precision arithmetic version of BLAS and LAPACK," Ver. 0.6.7, hdp://mplapack.sourceforge.net, 2010.
[11] F. Jézéquel and J.-M. Chesneaux, "CADNA: a library for estimating round-off error propagation," Computer Physics Communications 178(12), 2008.
[12] F. Févotte and B. Lathuilière, "Debugging and optimization of HPC programs in mixed precision with the Verrou tool," hal-02044101, 2019.
[13] D. Christophe et al., Verificarlo: Checking Floating Point Accuracy through Monte Carlo Arithmetic, ARITH2016.
[14] J. Vignes, "Discrete Stochastic Arithmetic for Validating Results of Numerical Software," Numer. Algorithms 37, 2004.
[15] Todd, R.: Introduction to Conditional Numerical Reproducibility (CNR), https://software.intel.com/en-us/articles/introduction-to-the-conditional- numerical-reproducibility-cnr, 2012.
[16] R. Iakymchuk et al., Reproducibility of Parallel Preconditioned Conjugate Gradient in Hybrid Programming Environments, IJHPCA 2020.
[17] F. Jézéquel et al., Can we avoid rounding-error estimation in HPC codes and still get trustful results?, NSV 20, 2020.
[18] T. Ogita et al., Accurate Sum and Dot Product, SIAM J. Sci. Comput., 2005.

Acknowledgements

This research was supported by the Japan Society for the Promotion of Science (JSPS) KAKENHI Grant #20KK0259. A part of this research used the FUJITSU Server PRIMERGY GX2570 (Wisteria/BDEC-01) at the Information Technology Center, the University of Tokyo (#jh220022).