Conjugate Gradient Solvers with Accuracy and Reproducibility Guarantees in Hybrid Parallel Environments

Roman Iakymchuk[1,2] and Daichi Mukunoki[3]
joint work with
Takeshi Ogita, Katsuhisa Ozaki, and Stef Graillat

[1] Sorbonne University, France
[2] Fraunhofer ITWM, Germany
[3] RIKEN Center for Computational Science, Japan
roman.iakymchuk@sorbonne-universite.fr

Sparse Days
Cerfacs, Toulouse
November 24th, 2020

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Motivation

Accuracy and Reproducibility of Preconditioned Conjugate Gradient

Iteration	Residual		
	MPFR	*Original* 1 core	*Original* 48 cores
0	0x1.19f179eb7f032p+49	0x1.19f179eb7f03**3**p+49	0x1.19f179eb7f03**3**p+49
2	0x1.f86089ece9f75p+38	0x1.f86089ece**5bd4**p+38	0x1.f86089ece**af76**p+38
9	0x1.fc59a29d329ffp+28	0x1.fc59a29d3**599a**p+28	0x1.fc59a29d32**d1b**p+28
10	0x1.74f5ccc211471p+22	0x1.74f5ccc**1d03cb**p+22	0x1.74f5ccc2**01246**p+22
...	...	...	...
40	0x1.7031058eb2e3ep-19	0x1.7031058**dd6bcf**p-19	0x1.7031058e**af4c2**p-19
42	0x1.4828f76bd68afp-23	0x1.4828f76**d1aa3**p-23	0x1.4828f76bd**a71a**p-23
45	0x1.8646260a70678p-26	0x1.8646260a**2dae8**p-26	0x1.8646260a**6da06**p-26
47	0x1.13fa97e2419c7p-33	0x1.13fa97e**1e76bf**p-33	0x1.13fa97e24**0f7c**p-33

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

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 till convergence with $tol = 10^{-8}$ for the gyro_k matrix from SuiteSparse

[a] R. Iakymchuk et al. Reproducibility Strategies for Parallel Preconditioned Conjugate Gradient. JCAM, 371, 2020, 112697

SORBONNE UNIVERSITÉ CRÉATEURS DE FUTURS DEPUIS 1257

Outline

1. ExBLAS: Exact BLAS
2. OzBLAS: Ozaki Scheme BLAS
3. Reproducible (Preconditioned) Conjugate Gradient
4. Conclusion and Future Work

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Computer arithmetic

Computer arithmetic approximates real numbers with their finite representations

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Computer arithmetic

Computer arithmetic approximates real numbers with their finite representations

Issues

- Floating-point arithmetic suffers from rounding errors
- Floating-point operations $(+, \times)$ are commutative but non-associative

$$(-1+1)+2^{-53} \neq -1+(1+2^{-53}) \quad \text{in double precision}$$

SORBONNE UNIVERSITÉ CRÉATEURS DE FUTURS DEPUIS 1257

Computer arithmetic

Computer arithmetic approximates real numbers with their finite representations

Issues

- Floating-point arithmetic suffers from rounding errors
- Floating-point operations $(+, \times)$ are commutative but non-associative

$$2^{-53} \neq 0 \quad \text{in double precision}$$

SORBONNE UNIVERSITÉ CRÉATEURS DE FUTURS DEPUIS 1257

Computer arithmetic

Computer arithmetic approximates real numbers with their finite representations

Issues

- Floating-point arithmetic suffers from rounding errors
- Floating-point operations ($+, \times$) are commutative but non-associative

$$(-1 + 1) + 2^{-53} \neq -1 + (1 + 2^{-53}) \quad \text{in double precision}$$

- Consequence: results of floating-point computations depend on the order of computation
- Results computed by performance-optimized parallel floating-point libraries may be often inconsistent

SORBONNE UNIVERSITÉ CRÉATEURS DE FUTURS DEPUIS 1257

Computer arithmetic

Computer arithmetic approximates real numbers with their finite representations

Issues

- Floating-point arithmetic suffers from rounding errors
- Floating-point operations ($+, \times$) are commutative but non-associative

$$(-1+1)+2^{-53} \neq -1+(1+2^{-53}) \quad \text{in double precision}$$

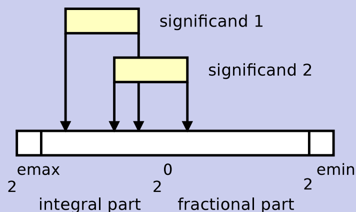
- Consequence: results of floating-point computations depend on the order of computation
- Results computed by performance-optimized parallel floating-point libraries may be often inconsistent

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

SORBONNE UNIVERSITÉ CRÉATEURS DE FUTURS DEPUIS 1257

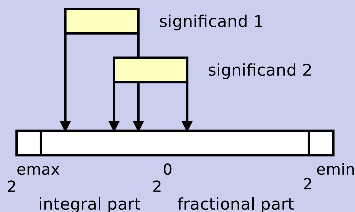
- “Infinite” precision: reproducible independently from the inputs

→ Example: Kulisch accumulator (=16 FLOPs)



- "Infinite" precision: reproducible independently from the inputs

→ Example: Kulisch accumulator (=16 FLOPs)



- Fixed FP Expansions (FPE) with Error-Free Transformations (EFT)

→ Example: double-double or quad-double (Briggs, Bailey, Hida, Li) (work well on a set of relatively close numbers)

Algorithm 1 (Dekker and Knuth)

Function$[r, s]$ = 2sum(a, b)

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

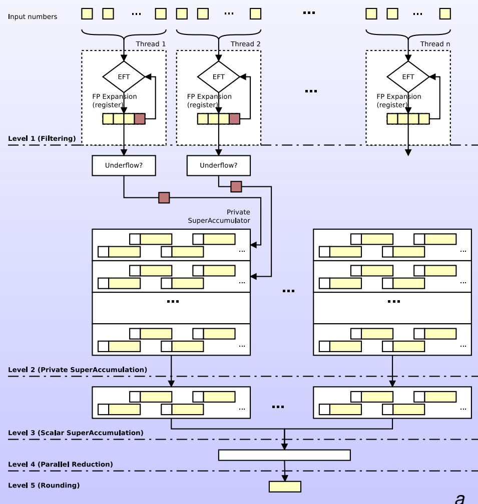
Algorithm 2 ($|a| \geq |b|$)

Function$[r, s]$ = fast2sum(a, b)

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

ExBLAS: Parallel Reduction

Highlights of the Algorithm



[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

Ozaki scheme (1/2)

Ozaki scheme: an accurate dot-product/matrix-multiplication method based on the error-free transformation of dot-product/matrix-multiplication[1]

For a dot-product,

1. Split the input vectors into several split vectors element-wisely (from higher to lower bits)
2. Compute the all-to-all product of those split vectors
3. Sum the results of the above all-to-all product

Analogy with a product of multi-digits numbers:

- $\boldsymbol{x} \cdot \boldsymbol{y} = (\boldsymbol{x}^{(1)} + \boldsymbol{x}^{(2)} + \boldsymbol{x}^{(3)}) \cdot (\boldsymbol{y}^{(1)} + \boldsymbol{y}^{(2)} + \boldsymbol{y}^{(3)})$
- $123 \times 456 = (100 + 20 + 3) \times (400 + 50 + 6)$

[1] K. Ozaki, T. Ogita, S. Oishi, and S. M. Rump. 2012. Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications. Numer. Algorithms 59, 1 (2012), 95–118.

SORBONNE UNIVERSITÉ CRÉATEURS DE FUTURS DEPUIS 1257

Ozaki scheme (2/2)

For $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^n$, where $\mathbb{F}$ is the set of floating-point numbers (FP64),

(1) Splitting:

$$\boldsymbol{x} = \sum_{p=1}^{s_x} \boldsymbol{x}_{\texttt{split}}{}^{(p)}, \quad \boldsymbol{y} = \sum_{q=1}^{s_y} \boldsymbol{y}_{\texttt{split}}{}^{(q)}, \quad \boldsymbol{x}_{\texttt{split}}{}^{(p)}, \boldsymbol{y}_{\texttt{split}}{}^{(q)} \in \mathbb{F}^n$$

- The splitting is performed so that those dot-products are error-free ($\texttt{fl}(\cdot)$ denote floating-point operations):

$$\left(\boldsymbol{x}_{\texttt{split}}{}^{(p)}\right)^T \boldsymbol{y}_{\texttt{split}}{}^{(q)} = \texttt{fl}\left(\left(\boldsymbol{x}_{\texttt{split}}{}^{(p)}\right)^T \boldsymbol{y}_{\texttt{split}}{}^{(q)}\right)$$

- These dot-products can be computed using the standard BLAS routine (DDOT)

(2) All-to-all product and **(3)** summation by NearSum[2]:

$$\boldsymbol{x}^T \boldsymbol{y} = \sum_{p=1}^{s_x} \sum_{q=1}^{s_y} \left(\boldsymbol{x}_{\texttt{split}}{}^{(p)}\right)^T \boldsymbol{y}_{\texttt{split}}{}^{(q)}$$

---

[2] S. Rump, T. Ogita, S. Oishi. Accurate Floating-Point Summation. TechReport. TU Hamburg. 2005

SORBONNE UNIVERSITÉ CRÉATEURS DE FUTURS DEPUIS 1257

Preconditioned Conjugate Gradient

Algorithm

$$\boxed{Ax = b}$$

Compute preconditioner for $A \rightarrow M$
Set starting guess $x^{(0)}$
Initialize $z^{(0)}, d^{(0)}, \beta^{(0)}, \tau^{(0)}, l := 0$ (iteration count)

$r^{(0)} := b - Ax^{(0)}$
$\tau^{0} := < r^{(0)}, r^{(0)} >$
while $(\tau^{(l)} > \tau_{\max})$

Step	Operation	Kernel
$S1:$	$w^{(l)} := Ad^{(l)}$	SpMV
$S2:$	$\rho^{(l)} := \beta^{(l)} / < d^{(l)}, w^{(l)} >$	DOT product
$S3:$	$x^{(l+1)} := x^{(l)} + \rho^{(l)} d^{(l)}$	AXPY
$S4:$	$r^{(l+1)} := r^{(l)} - \rho^{(l)} w^{(l)}$	AXPY
$S5:$	$z^{(l+1)} := M^{-1} r^{(l+1)}$	Apply precond.
$S6:$	$\beta^{(l+1)} := < z^{(l+1)}, r^{(l+1)} >$	DOT product
$S7:$	$d^{(l+1)} := (\beta^{(l+1)} / \beta^{(l)}) d^{(l)} + z^{(l+1)}$	AXPY-like
$S8:$	$\tau^{(l+1)} := < r^{(l+1)}, r^{(l+1)} >$	DOT product
	$l := l + 1$	

end while

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

- Identify sources of non-reproducibility:
  - `spmv`
  - `dot`
  - `axpy`

Solutions

- **Combine arithmetic solutions, reorganization of operations, and sequential executions**
  → aiming for lighter or lightweight approaches
- `axpy` is reproducible thanks to `fma`
- `spmv` computes blocks of rows in parallel, but with $a * b + / - c * d$
  → ensure deterministic execution with explicit `fma`
- accurate and reproducible `dot`
  → ExBLAS-based approach
  → FPE with size 8 and early-exit
  → OzBLAS-based approach

SORBONNE UNIVERSITÉ CRÉATEURS DE FUTURS DEPUIS 1257

dot product with ExBLAS/ FPEs

- `exdot` is virtually built upon `exsum` and `twoprod`

→ `twoprod`(a,b) (3 FLOPs):

1: $res \leftarrow a \cdot b,$
2: $err \leftarrow \texttt{fma}(a, b, -res)$

Algorithm 3: Distributed DOT product of vectors *a* and *b* with FPEs.

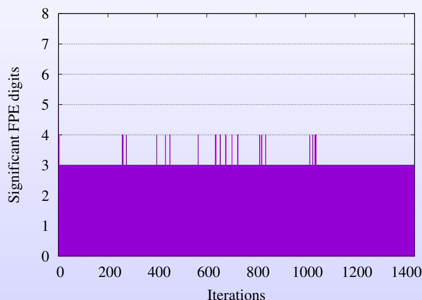
Function $dot(N, a, b, fpe, fperr)$
  local DOT product with subvectors of size N_k
  for $i = 0 \rightarrow N_k - 1$ **do**
    res = `twoprod`($a[i], b[i], err$)
    ExpansionAccumulate(fpe, res);
    ExpansionAccumulate($fperr, err$);
  end
  Merge FPEs with ExpansionAccumulate(a, x)
  MPI reduction of FPEs
  Rounding to the target format

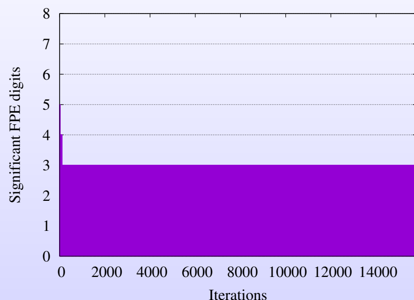
Algorithm 4: Adding a floating-point number *x* to a floating-point expansion *a* of size *p*.

Function ExpansionAccumulate(a, x)
  Input: x is a floating-point number.
  Output: a is a FPE containing the result.
  for $i = 0 \rightarrow p - 1$ **do**
    $(a[i], x)$:= `twosum`($a[i], x$)
  end

dot product with ExBLAS/ FPEs

Required precision





msc01050 (Boeing)

$NNZ = 26,198$

$cond(A) = 9.0e + 15$

gyro_k (Oberwolfach)

$NNZ = 1,021,159$

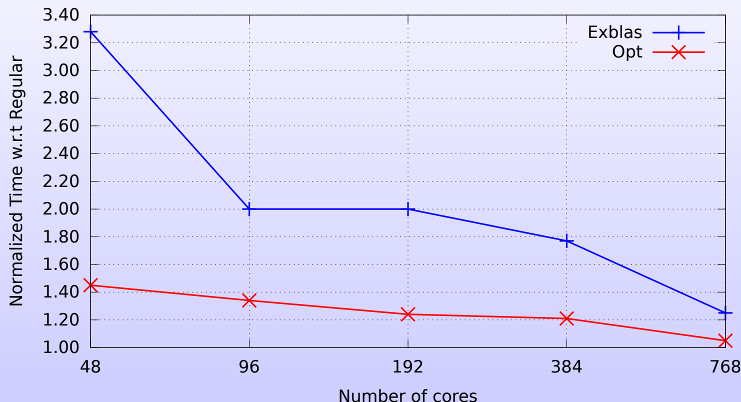
$cond(A) = 1.1e + 09$

"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.

SORBONNE UNIVERSITÉ CRÉATEURS DE FUTURS DEPUIS 1257

Reproducible PCG

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



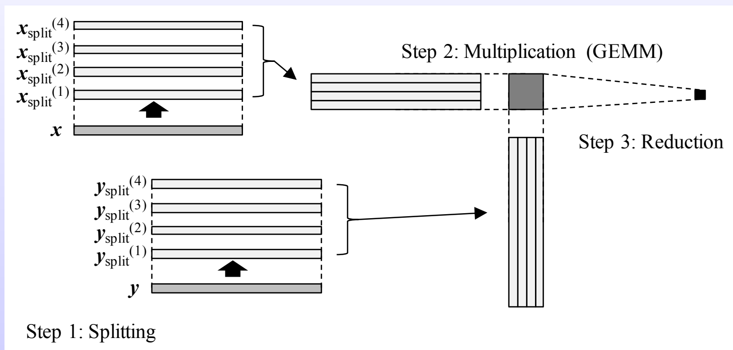
MN4 nodes: 2x24-core Intel Xeon Platinum 8160 w Intel Omni-Path

Opt is FPE with 8 digits and early-exit

SORBONNE UNIVERSITÉ CRÉATEURS DE FUTURS DEPUIS 1257

dot product with Ozaki scheme

Using GEMM



Dot-product with Ozaki scheme (when the number of split vectors is 4).[3]

[3] Mukunoki et al. Conjugate Gradient Solvers with High Accuracy and Bit-wise Reproducibility between CPU and GPU using Ozaki scheme. HPC-Asia 2021

- 2 CPUs: Intel Xeon Gold 6126 (Skylake, 24 cores)
  Intel Xeon Phi 7250 (KNL, 68 cores)
- $\boldsymbol{b} = \boldsymbol{x}_0 = (1, 1, ..., 1)^T$
- Iteration is terminated at $||\boldsymbol{r}_i||/||\boldsymbol{b}|| \leq 10^{-16}$
- 8 matrices from SuiteSparse Matrix Collection

Test matrices (S.P.D., $n \times n$ with nnz non-zeros, sorted by nnz/n)

#	Matrix	n	nnz	nnz/n	type
1	tmt_sym	5,080,961	726,713	7.0	electromagnetics problem
2	gridgena	48,962	512,084	10.5	optimization problem
3	cfd1	1,825,580	70,656	25.8	computational fluid dynamics problem
4	cbuckle	13,681	676,515	49.4	structural problem
5	BenElechi1	245,874	13,150,496	53.5	2D/3D problem
6	gyro_k	17,361	1,021,159	58.8	duplicate model reduction problem
7	pdb1HYS	36,417	4,344,765	119.3	weighted undirected graph
8	nd24k	72,000	28,715,634	398.8	2D/3D problem

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Comparison of both solutions

Overheads of the ExBLAS-based (FP64Ex-CR) and OzBLAS-based (FP64Oz-CR) v.s. FP64 (lowest overhead); $tol = 10^{-16}$

#	Matrix	FP64Ex-CR		FP64Oz-CR[a]			
		Skylake	KNL	Skylake	KNL	GPU1	GPU2
1	tmt_sym	54.9	89.5	28.2	36.6	17.1	21.7
2	gridgena	16.9	13.8	11.6	10.8	12.6	12.3
3	cfd1	22.3	17.0	15.1	11.7	12.3	14.9
4	cbuckle	11.1	6.4	16.0	11.5	9.7	10.3
5	BenElechi1	15.8	41.2	16.7	18.3	12.8	16.9
6	gyro_k	9.5	7.5	12.7	10.1	7.6	8.5
7	pdb1HYS	9.3	12.6	13.1	8.1	6.0	6.9
8	nd24k	5.0	22.1	13.8	12.1	5.5	6.4

[a] NB: Results are for CG w/o preconditioner. ExBLAS-based and OzBLAS-based are different implementations. OzBLAS-based is initiated from FP64.

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Conclusion and Future Work

Conclusion

- Combined **algorithmic and programming strategies** led to reproducible Conjugate Gradient on CPUs and GPUs

→ All inner-product-based operations are correctly-rounded

- **ExBLAS- and FPE-based PCG** showed **20 % and 5 % overhead** on up to 764 cores but requires development of BLAS routines
- **Ozaki-scheme-based CG is tunable and efficient** as it relies on vendor implementations of BLAS
- ExBLAS-like PCG: `github.com/riakymch/ReproCG` and `github.com/riakymch/ReproCG_MPI_OMP`
- Ozaki scheme CG: `http://www.math.twcu.ac.jp/ogita/post-k/results.html`

Future Work

- Accuracy tuning with the Ozaki scheme
- Floating-point expansions with few digits plus error checker
- Different preconditioners and pipelined PCGs