Hierarchical and modular approach for reproducible and accurate linear algebra algorithms

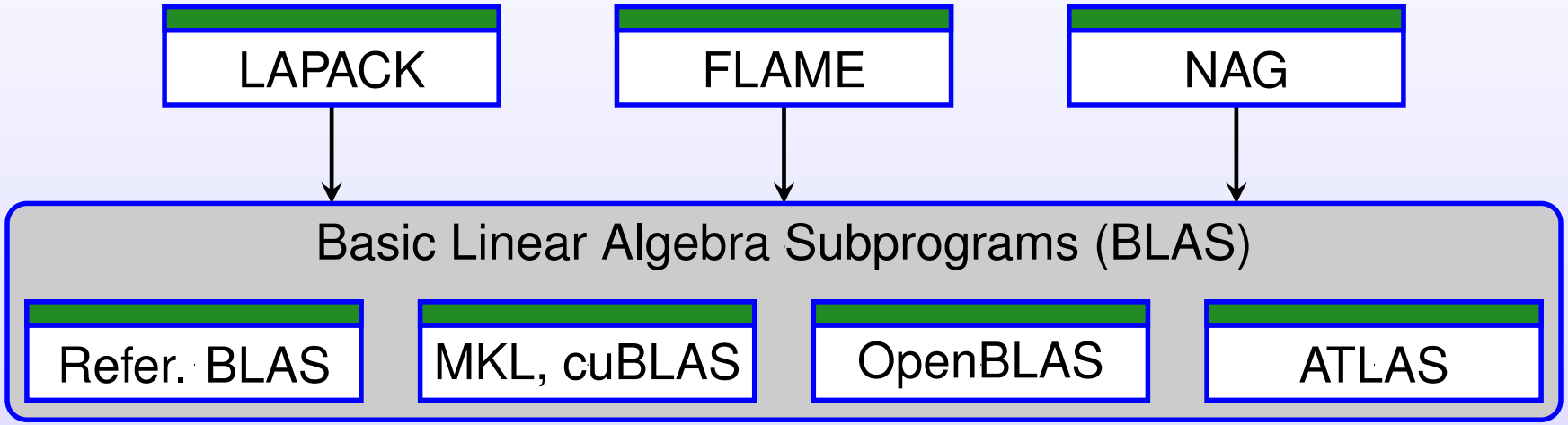
Roman Iakymchuk[1,2]
joint work with
Maria Barreda[3], José I. Aliaga Estellés[2], Enrique S. Quintana-Ortí[4], and Stef Graillat[1]

[1]Sorbonne University, France
[2]Fraunhofer ITWM, Germany
[3]Univesitat Jaime I, Spain
[4]Universitat Politècnica de València, Spain
roman.iakymchuk@sorbonne-universite.fr

Workshop on Large-scale Parallel Numerical Computing Technology – HPC and Computer Arithmetics toward Minimal-Precision Computing
R-CCS, Kobe, Japan
Jan 29th-30th, 2020

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Linear Algebra Libraries



$\Downarrow$

BLAS-1 [1979]:	$y := y + \alpha x$ $\alpha := \alpha + x^T y$	$\alpha \in \mathbb{R}; x, y \in \mathbb{R}^n$	$2/3$
BLAS-2 [1988]:	$A := A + xy^T$ $y := A^{-1}x$	$A \in \mathbb{R}^{n \times n}; x, y \in \mathbb{R}^n$	2
BLAS-3 [1990]:	$C := C + AB$ $C := A^{-1}B$	$A, B, C \in \mathbb{R}^{n \times n}$	$n/2$

Research goals

- Compute BLAS operations with floating-point numbers fast and precise, ensuring their numerical reproducibility, on a wide range of architectures
- **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

ExBLAS – **Ex**act **BLAS**

- **Ex**BLAS-1: `ExSUM`, `ExSCAL`, `ExDOT`, `ExAXPY`, ...
- **Ex**BLAS-2: `ExGER`, `ExGEMV`, `ExTRSV`, `ExSYR`, ...
- **Ex**BLAS-3: `ExGEMM`, `ExTRSM`, `ExSYR2K`, ...

- Construct reproducible higher-level operations like matrix factorizations and **iterative solvers** using ExBLAS

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Outline

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Outline

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Background: Computer arithmetic

Computer arithmetic

Approximate real numbers by numbers that have a finite representation

Background: Computer arithmetic

Computer arithmetic

Approximate real numbers by numbers that have a finite representation

Problems

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

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

Background: Computer arithmetic

Computer arithmetic

Approximate real numbers by numbers that have a finite representation

Problems

- 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

Background: Computer arithmetic

Computer arithmetic

Approximate real numbers by numbers that have a finite representation

Problems

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

  $(-1+1)+2^{-53} \neq -1+(1+2^{-53})$ 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: each run returns a different result

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Background: Reproducibility

Sources of Non-Reproducibility

- Changing Data Layouts:
  - Data partitioning
  - Data alignment
- Changing Hardware Resources
  - Number of threads
  - Fused Multiply-Add support: $a \cdot b + c$
  - Intermediate precision (64 bits, 80 bits, 128 bits, etc)
  - Data path (SSE, AVX, GPU warp, etc)
  - Number of processors
  - Network topology

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Outline

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Existing Solutions

- **Fix the Order of Computations**
  - Sequential mode: intolerably costly at large-scale systems
  - Fixed reduction trees: substantial communication overhead

  $\rightarrow$ Example: Intel **C**onditional **N**umerical **R**eproducibility in MKL ($\sim 2x$ for datum, no accuracy guarantees)

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Existing Solutions

- **Fix the Order of Computations**
  - Sequential mode: intolerably costly at large-scale systems
  - Fixed reduction trees: substantial communication overhead

  → Example: Intel **C**onditional **N**umerical **R**eproducibility in MKL ($\sim 2x$ for datum, no accuracy guarantees)

- **Eliminate/Reduce the Rounding Errors**
  - Fixed-point arithmetic: limited range of values
  - Fixed FP expansions with Error-Free Transformations (EFT)

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

  - "Infinite" precision: reproducible independently from the inputs

  → Example: Kulisch accumulator (considered inefficient)

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Existing Solutions

- **Fix the Order of Computations**
  - Sequential mode: intolerably costly at large-scale systems
  - Fixed reduction trees: substantial communication overhead

  → Example: Intel **C**onditional **N**umerical **R**eproducibility in MKL ($\sim 2x$ for datum, no accuracy guarantees)

- **Eliminate/Reduce the Rounding Errors**
  - Fixed-point arithmetic: limited range of values
  - Fixed FP expansions with Error-Free Transformations (EFT)

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

  - "Infinite" precision: reproducible independently from the inputs

  → Example: Kulisch accumulator (considered inefficient)

- **Libraries**
  - ExBLAS: Exact BLAS (Iakymchuk et al.)
  - ReproBLAS: Reproducible BLAS (Demmel et al.)
  - RARE-BLAS: Repr. Acc. Rounded and Eff. BLAS (Chohra et al.)
  - OzBLAS: Ozaki-scheme BLAS (Mukunoki et al.)

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Exact Multi-Level Parallel Reduction

Preliminaries

- Fixed FP expansions (FPE) with Error-Free Transformations

→ 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$[r, s]$ = `twosum`(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$[r, s]$ = `twosum`(a, b)

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

Exact Multi-Level Parallel Reduction

Preliminaries

- Fixed FP expansions (FPE) with Error-Free Transformations

→ 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$[r, s]$ = `twosum`(a, b)

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

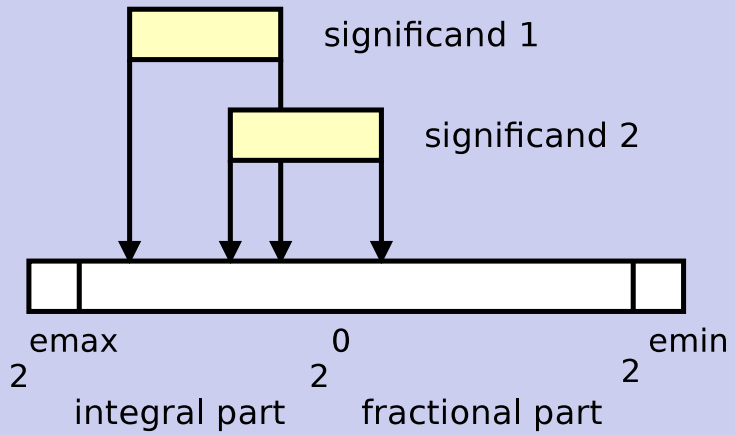
Algorithm 2 ($|a| \geq |b|$)

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

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

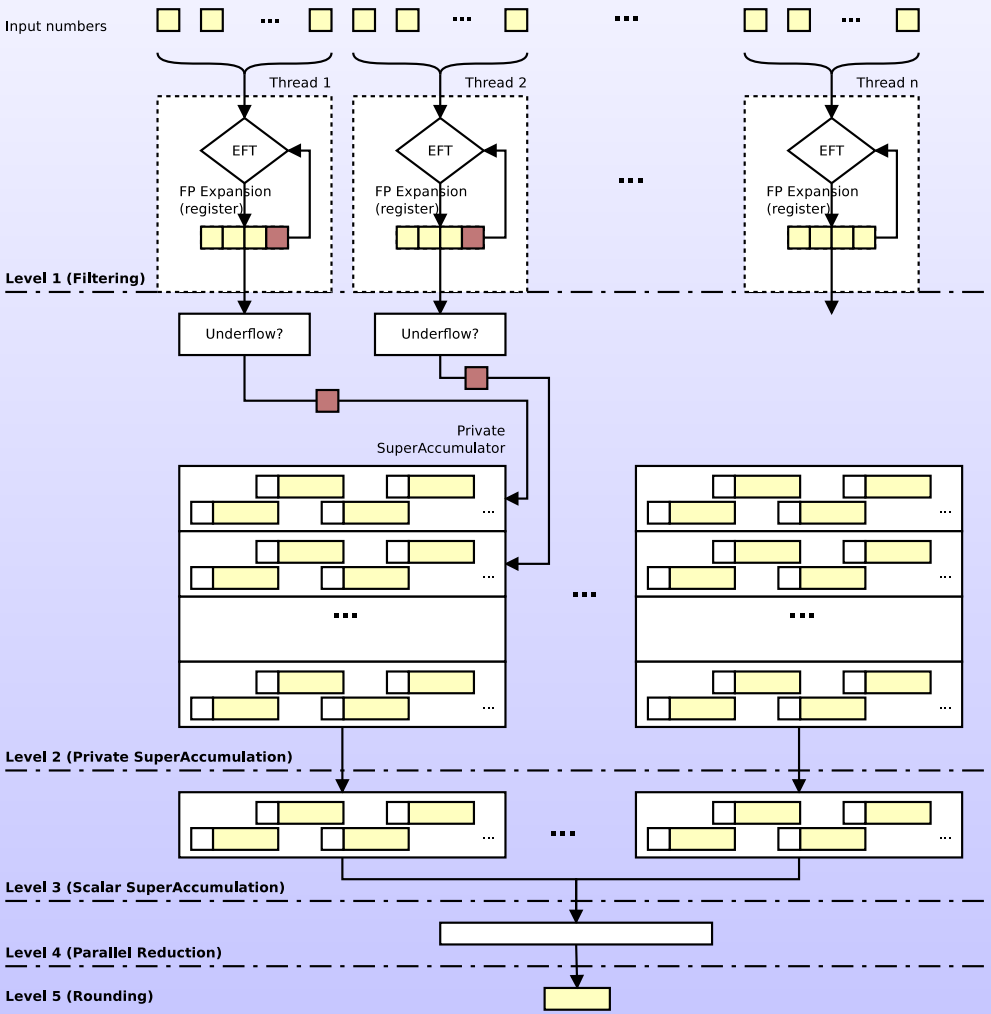
- "Infinite" precision: reproducible independently from the inputs

→ Example: Kulisch accumulator (=16 FLOPs)



Exact Multi-Level Parallel Reduction

Highlights of the Algorithm

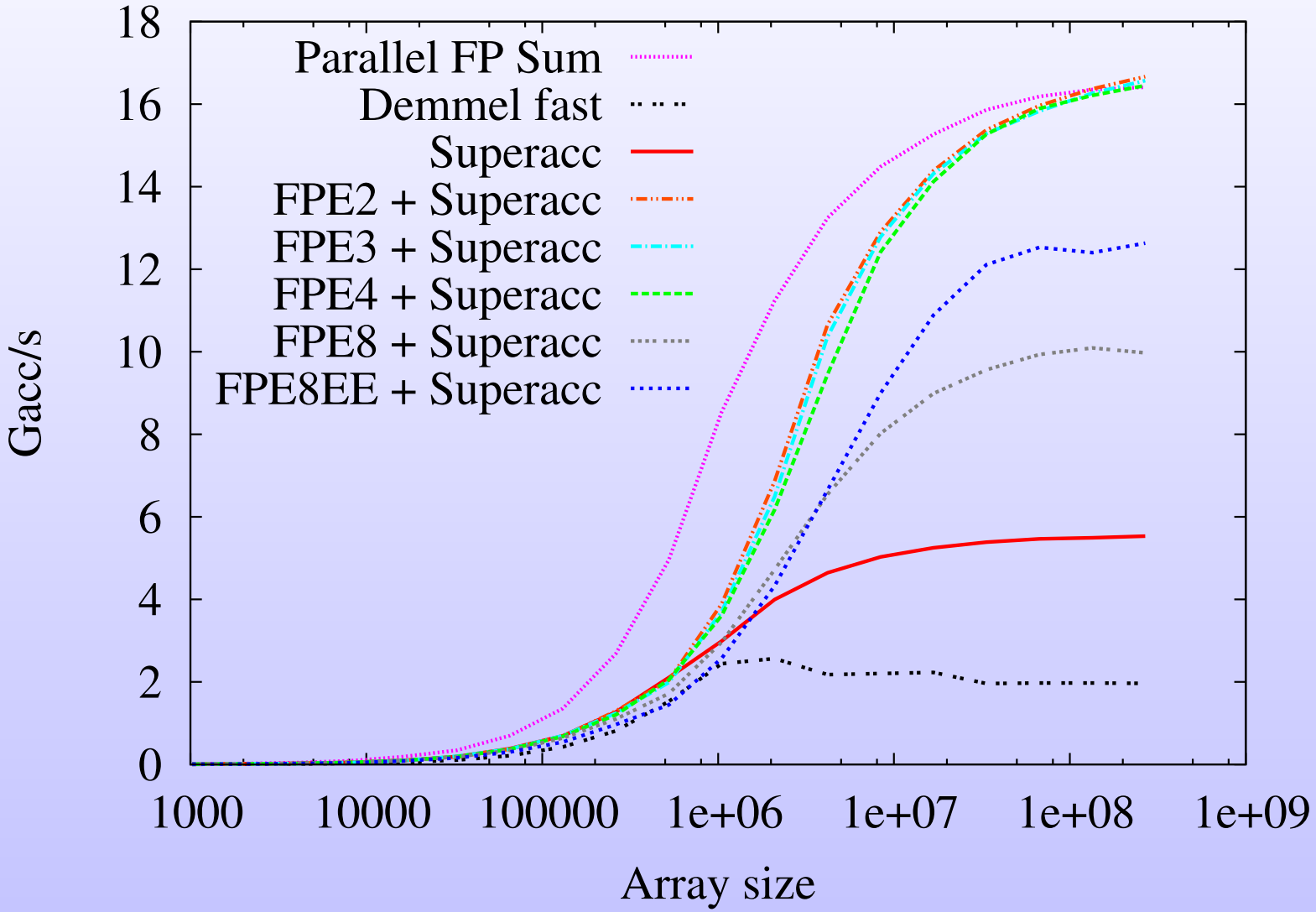


- Parallel algorithm with 5-levels
- Suitable for today's parallel architectures
- Based on FPE with EFT and Kulisch accumulator
- Guarantees "inf" precision

→ **bit-wise reproducibility**

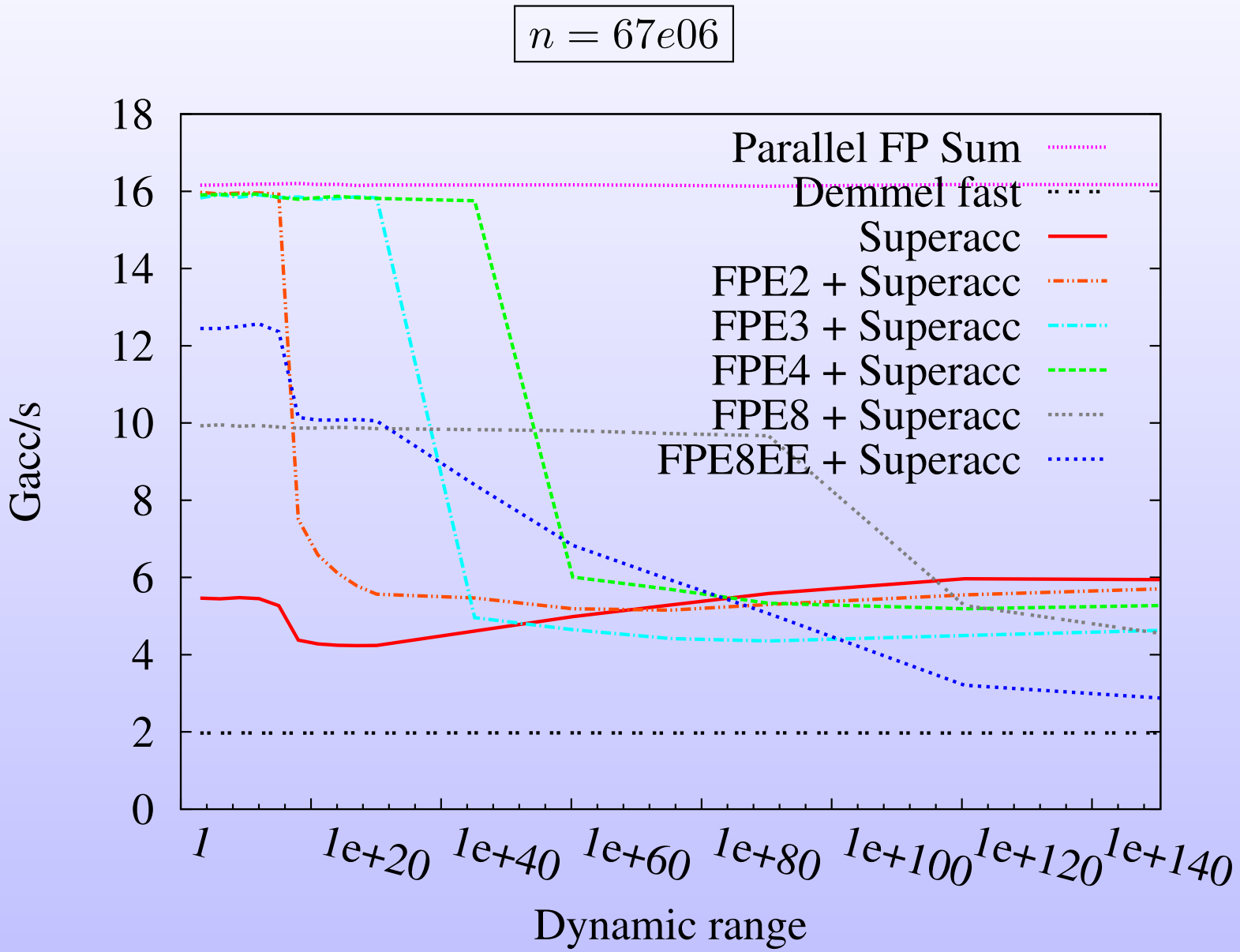
ExSUM: Results

Performance Scaling on NVIDIA Tesla K20c



ExSUM: Results

Data-Dependent Performance on NVIDIA Tesla K20c



ExBLAS-1 Highlights (1/2)

BLAS-1 routines

- Some are virtually built upon `exsum`

→ For instance, `exdot` = `twoprod` + `exsum`

→ `twoprod`(a,b) (= 3 FLOPs):

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

- The others are trivial: `exaxpy` = $\texttt{fma}(\alpha, x[i], y[i])$

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

ExBLAS-1 Highlights (1/2)

BLAS-1 routines

- Some are virtually built upon `exsum`

→ For instance, `exdot` = `twoprod` + `exsum`

→ `twoprod`(a,b) (= 3 FLOPs):

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

- The others are trivial: `exaxpy` = $\texttt{fma}(\alpha, x[i], y[i])$

`exscal`

- $x := \alpha \cdot x \rightarrow$ correctly rounded and reproducible
- Within LU: $x := 1/\alpha \cdot x \rightarrow$ not correctly rounded
- `exinvscal`: $x := x/\alpha \rightarrow$ correctly rounded and reproducible

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

ExBLAS-1 Highlights (1/2)

BLAS-1 routines

- Some are virtually built upon `exsum`

→ For instance, `exdot` = `twoprod` + `exsum`

→ `twoprod`(a,b) (= 3 FLOPs):

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

- The others are trivial: `exaxpy` = $\texttt{fma}(\alpha, x[i], y[i])$

`exscal`

- $x := \alpha \cdot x \rightarrow$ correctly rounded and reproducible
- Within LU: $x := 1/\alpha \cdot x \rightarrow$ not correctly rounded
- `exinvscal`: $x := x/\alpha \rightarrow$ correctly rounded and reproducible

`exger`

- General case: $A := \alpha \cdot x \cdot y^T + A$
- Within LU ($\alpha = 1.0$): $A := x \cdot y^T + A$.
  Using `fma` → correctly rounded and reproducible

ExBLAS-1 Highlights (2/2)

Programming and compilation solutions

axpy

- $y := \alpha \cdot x + y$

$\rightarrow$ $\texttt{fma}(\alpha, x[i], y[i])$

$\rightarrow$ correctly rounded and reproducible

axpy-like

- $y := \alpha \cdot x + \beta \cdot y$
- Warning: C++ compilers can change the execution order

$\rightarrow$ instruct compiler to use `fma`, eg `std::fma` with `C++11`

$\rightarrow$ prevent the use of value changing optimization techniques, eg `-fp-model precise` for `icc`

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

An unblocked LU Factorization

Variant 5

LU Factorization

$$\pi_1 := PivIndex\left(\frac{\alpha_{11}}{a_{21}}\right) \quad (\textbf{max})$$

$$\left(\frac{\alpha_{11}}{a_{21}}\right) := P(\pi_1)\left(\frac{\alpha_{11}}{a_{21}}\right) \quad (\textbf{swap})$$

$$a_{21} := a_{21}/\alpha_{11} \quad (\textbf{scal})$$

$$A_{22} := A_{22} - a_{21}a_{12}^T \quad (\textbf{ger})$$

	i	1	p
i	A_{00}	a_{01}	A_{02}
1	a_{10}^T	α_{11}	a_{12}^T
p	A_{20}	a_{21}	A_{22}

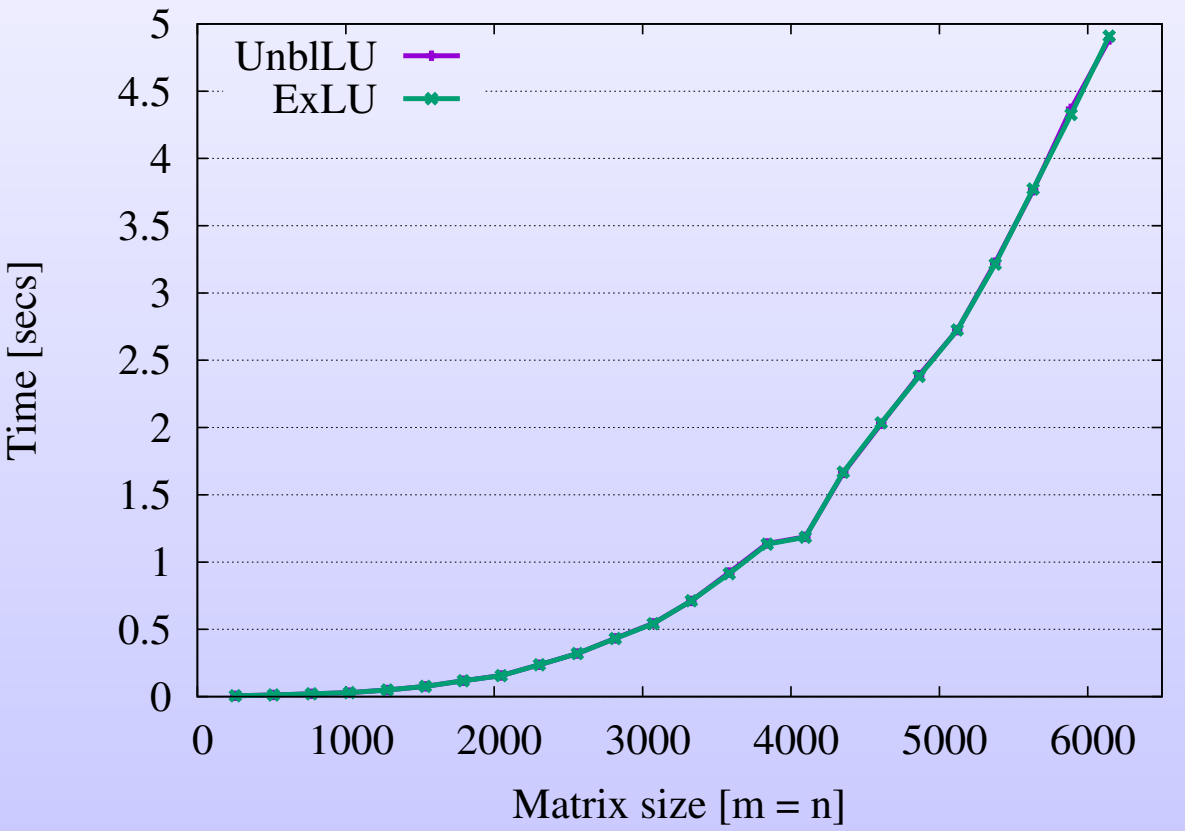
3×3 partitioning of A

- **max** is reproducible once the choice among equal elements is deterministic

An unblocked LU Factorization

Performance Scaling on NVIDIA Pascal P100

$$A = LU$$



variant 5 of LU

```
max()
swap()
```

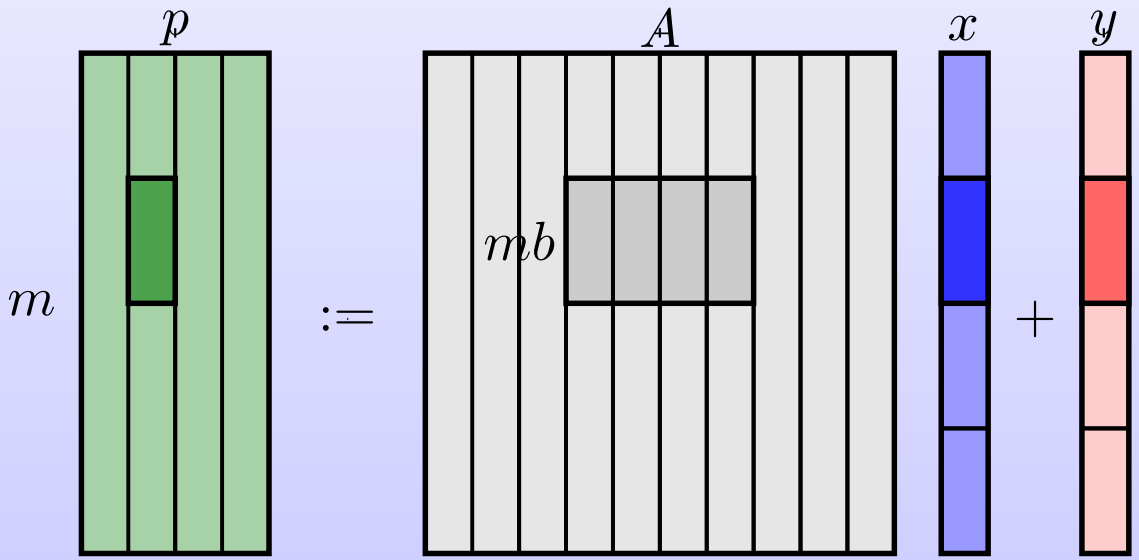
$a_{21} := a_{21}/\alpha_{11}$ `scal`

$A_{22} := A_{22} - a_{21}a_{12}^T$ `ger`

ExBLAS-2 Highlights

Matrix-Vector Product

gemv: $y := \alpha Ax + \beta y$

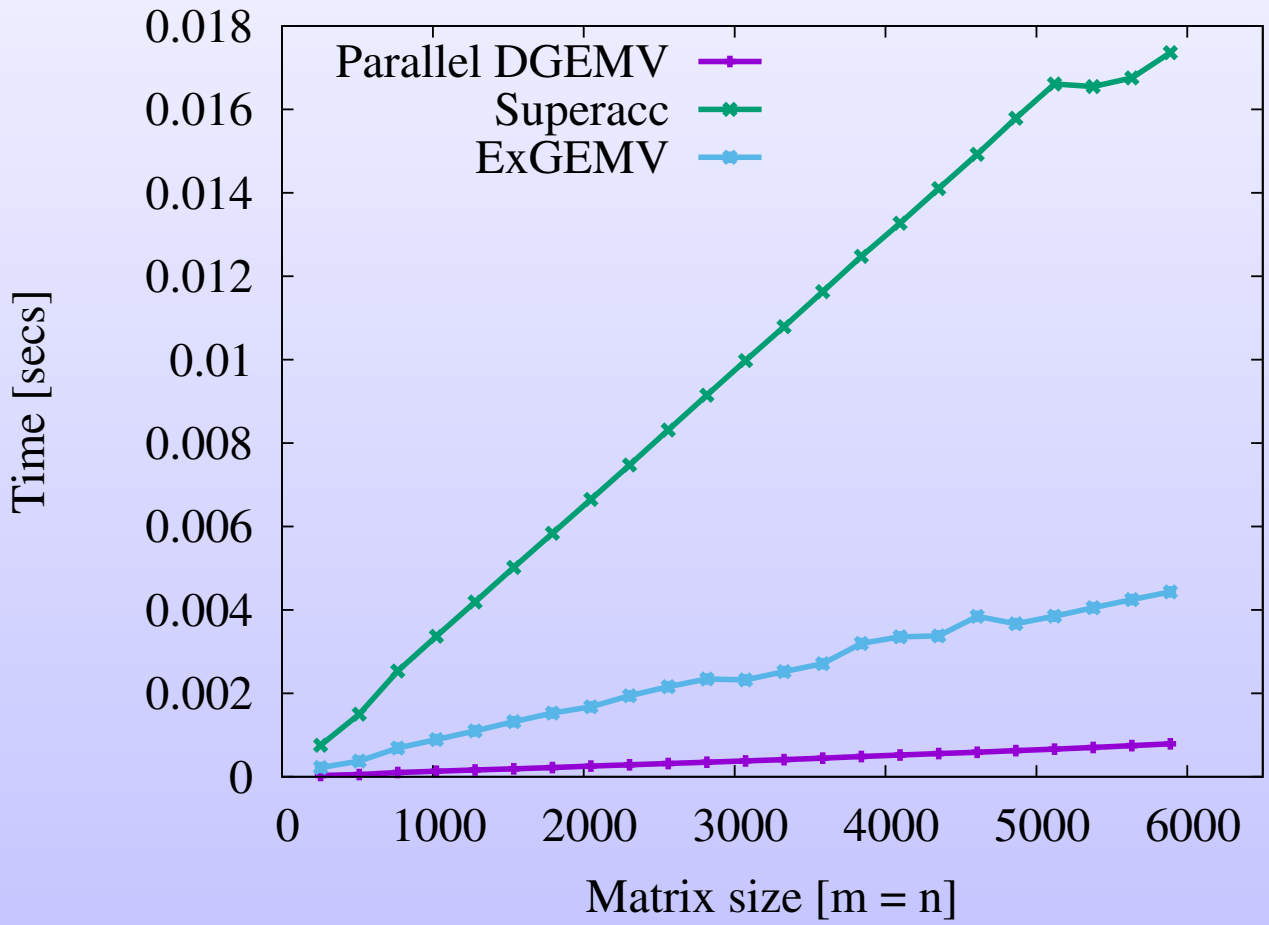


- Based on `exdot`
- `twoprod`(a, b)
  1: $r \leftarrow a \cdot b$
  2: $s \leftarrow \texttt{fma}(a, b, -r)$
- $\texttt{fma}(a, b, c) = a * b + c$

Matrix-Vector Product

Performance Scaling on NVIDIA Pascal P100

gemv: $y := Ax + y$

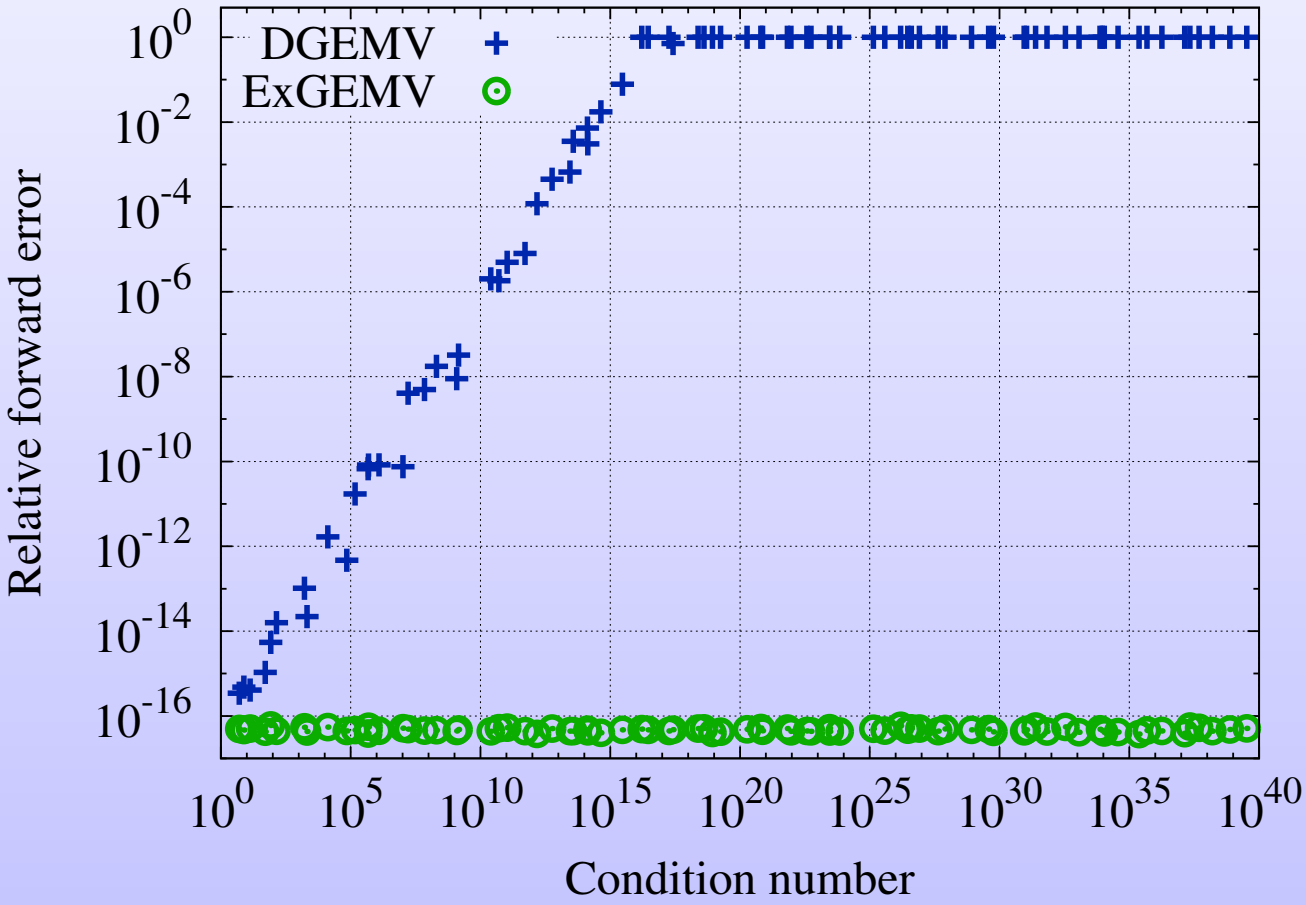


- Blocked `exgemv`
- Based on `exdot`

Matrix-Vector Product

Accuracy

gemv: $y := Ax$



- Preserve every bit of information
- Correctly-rounded
- $\mathrm{cond}(A, x) = \frac{\||A|\cdot|x|\|}{\|A\cdot x\|}$

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Outline

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Preconditioned Conjugate Gradient

Problem

We consider the **efficient** solution of linear system

$$Ax = b,$$

where

- $A \in \mathbb{R}^{n \times n}$ is a **large** and **sparse** symmetric positive definite (SPD) coefficient matrix
- $x \in \mathbb{R}^{n \times n}$ is a sought-after solution
- $b \in \mathbb{R}^{n \times n}$ is a give right-hand side vector

Solution

We propose to address $Ax = b$ **iteratively** using

- Preconditioned Conjugate Gradient (PCG) method
  - Among the most often used iterative approaches to solve SPD linear systems
  - Jacobi preconditioner is good enough for many problems
- On clusters of multicore processors with Message Passing Interface

Preconditioned Conjugate Gradient

Algorithm

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

Preconditioned Conjugate Gradient

Communication

Compute preconditioner for $A \rightarrow M$
Set starting guess x
Initialize $z, d, \beta, \tau, l := 0$

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

Step	Operation	Communication
	$\beta' := \beta$	–
$S1:$		
$S1.1:$	$d \rightarrow e$	Allgatherv
$S1.2:$	$w := Ae$	–
$S2:$	$\rho := \beta / < d, w >$	Allreduce
$S3:$	$x := x + \rho d$	–
$S4:$	$r := r - \rho w$	–
$S5:$	$z := M^{-1} r$	–
$S6:$	$\beta := < z, r >$	Allreduce
$S8:$	$\tau := < r, r >$	Allreduce
$S7:$	$d := (\beta / \beta')d + z$	–
	$l := l + 1$	

end while

Reproducible Conjugate Gradient

Overview of reproducibility strategies

Sources

- Identify sources of non-reproducibility: `dot` (parallel reduction), `axpy`, and `spmv`

Solutions

- **Combine sequential executions, reorganization of operations, and arithmetic solutions**

$\rightarrow$ aiming for lighter or lightweight approaches

- `axpy` is made reproducible thanks to `fma`
- `spmv` computes blocks of rows in parallel, but with $a * b + / - c * d$

$\rightarrow$ ensure deterministic execution with explicit `fma`

- `dot` –> apply the ExBLAS- and FPE-based approaches

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Distributed Dot Product with ExBLAS

- Exploit the ExBLAS parallel reduction with the `twoprod` EFT
- Drawbacks:
  - The required memory storage
  - The number of required operations

Distributed Dot Product with ExBLAS

- Exploit the ExBLAS parallel reduction with the `twoprod` EFT
- Drawbacks:
  - The required memory storage
  - The number of required operations

Distributed Dot Product with FPEs

- The PCG method can accommodate accurate and reproducible computations using few floating point numbers (FPEs)
- FPE8 is capable to represent 8x53 bits of significant. FPE3 is frequently enough
- Combined with the early-exit technique
- Improves algorithm's performance and enhance its accuracy

Reproducible Conjugate Gradient

Reproducible dot product and allreduce with ExBLAS

Dot product

- Extended ExSUM to ExDOT using `twoprod` EFT
- This is combined with reduction among all processes

Allreduce

- Allreduce is split into Reduce and Bcast
- This facilitates implementation but also delivers better performance for some cases

Listing 1: Reproducible Allreduce with ExBLAS.

```
1 std::vector<int64_t> h_superacc(BIN_COUNT);
2 exblas::exdot (..., &h_superacc[0]);
3 exblas::Normalize (&h_superacc[0]);
4 MPI_Reduce (&h_superacc[0],..., BIN_COUNT,
      MPI_LONG, MPI_SUM);
5 if (myId == 0) {
6   beta = exblas::Round (&h_superacc[0]);
7 }
8 MPI_Bcast (&beta, 1, MPI_DOUBLE, ...);
```

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Reproducible Conjugate Gradient

Reproducible dot product with FPEs

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**
  - $res = \texttt{twoprod}(a[i], b[i], err)$
  - 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**
  - $(a[i], x) := \texttt{twosum}(a[i], x)$
- **end**

Reproducible Conjugate Gradient

Reproducible allreduce with FPEs

Listing 2: Reproducible Allreduce with FPEs only.

```
1  std::vector<double> fpe(N);
2  dot (..., &fpe[0]);
3  renormalize(&fpe[0]);     // optional
4  MPI_Op Op;     // user-defined reduction operation
5  MPI_Op_create (fpesum, 1, &Op);
6  MPI_Reduce (&fpe[0], ..., N, MPI_DOUBLE, Op);
7  if (myId == 0) {
8    beta = Round (&fpe[0]);    // Add3
9  }
10 MPI_Bcast (&beta, 1, MPI_DOUBLE, ...);
```

Algorithm 5: Aggregation of two FPEs of size p.

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

Reproducible Conjugate Gradient

Rounding FPEs

Listing 3: Rounding a FPE to double using the Add3 algorithm.

```
1  inline static T Round( const T *fpe ) {
2      union {
3          T d;
4          int64_t l;
5      } thdb;
6      T tl;
7      T th = twosum(fpe[1], fpe[2], tl);
8      if (tl != 0.0) {
9          thdb.d = th;
10         // if the mantissa of th is odd, we are done
11         if (!(thdb.l & 1)) {
12             // choose the rounding direction
13             // depending of the signs of th and tl
14             if ((tl > 0.0) ^ (th < 0.0))
15                 thdb.l++;
16             else
17                 thdb.l--;
18             th = thdb.d;
19         }
20     }
21     // final addition rounded to nearest
22     return fpe[0] + th;
23 }
```

For FPEs of size eight, we rely upon `NearSum`[a]

[a] S. M. Rump, T. Ogita, S. Oishi, Accurate floating-point summation part ii: Sign, k-fold faithful and rounding to nearest, SIAM J. Sci. Comput. 31 (2008) 1269-1302.

Test matrices

- Sparse positive definite coefficient matrix
- 3D Poison's equation with 27 stencil points
- Transform it into a matrix band $\rightarrow$ The size of the band depends on the number of nodes (100 x #nodes)
- N = 4,000,000 rows/columns, but increase its bandwidth proportionally to the hardware resources

Set-up

- Two versions of reproducible PCG: ExBLAS- and FPE-based (Opt)
- Two different clusters: Tintorrum and Marenostrum4
- Reproducibility results of residual and direct error
- Strong and weak scaling
  - Strong scaling: Fix the matrix size to N = 16,000,000 and band_size = 100 and increase the number of cores
  - Weak scaling: Fix the matrix size to N = 4,000,000 and increase band_size from 100 to 100 x max_nodes

Reproducible Conjugate Gradient

Accuracy and reproducibility results

Iteration	Residual			
	MPFR	*Original* 1 proc	*Original* 48 procs	*Exblas & Opt*
0	0x1.19f179eb7f032p+49	0x1.19f179eb7f03**3**p+49	0x1.19f179eb7f03**3**p+49	0x1.19f179eb7f032p+49
2	0x1.f86089ece9f75p+38	0x1.f86089**f08810d**p+38	0x1.f86089e**d07a76**p+38	0x1.f86089ece9f75p+38
9	0x1.fc59a29d329ffp+28	0x1.fc59a29d**1b6a**p+28	0x1.fc59a29d**2e989**p+28	0x1.fc59a29d329ffp+28
10	0x1.74f5ccc211471p+22	0x1.74f5cc**b8203ad**p+22	0x1.74f5ccc**1fafef**p+22	0x1.74f5ccc211471p+22
...	...	...	...	...
40	0x1.7031058eb2e3ep-19	0x1.703105**aea0e8a**p-19	0x1.7031058e**8ff5a**p-19	0x1.7031058eb2e3ep-19
42	0x1.4828f76bd68afp-23	0x1.4828f**6fabbf2a**p-23	0x1.4828f76b**b9038**p-23	0x1.4828f76bd68afp-23
45	0x1.8646260a70678p-26	0x1.864626**01300d2**p-26	0x1.8646260a7**1301**p-26	0x1.8646260a70678p-26
47	0x1.13fa97e2419c7p-33	0x1.13fa9**98038c44e**p-33	0x1.13fa97e**54e903**p-33	0x1.13fa97e2419c7p-33

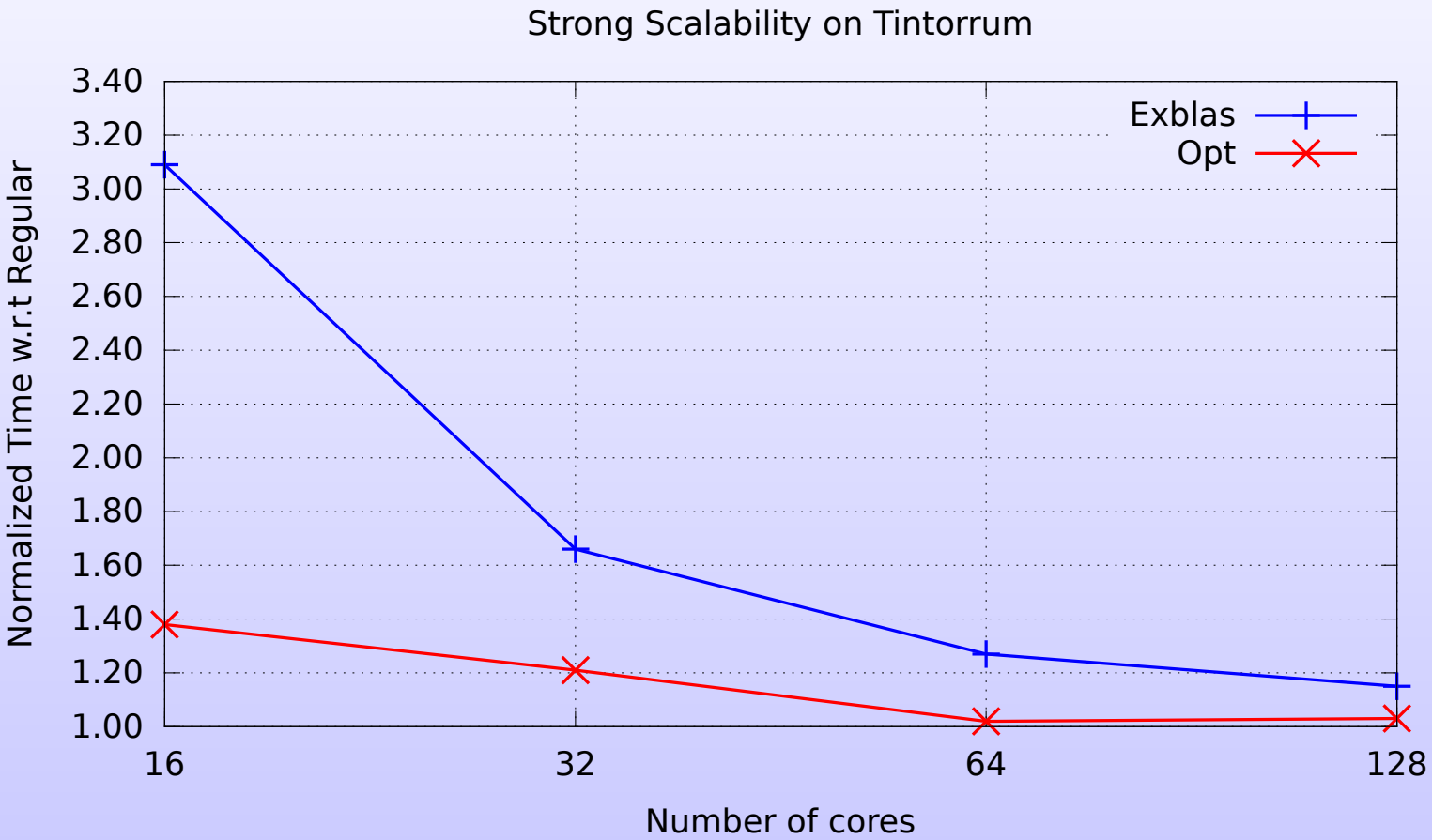
Table 3: Accuracy and reproducibility comparison on the intermediate and final residual against MPFR for a matrix with condition number of 10^{12}. The matrix is generated following the procedure from Section 5.1 with n=4,019,679 (159^3).

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Reproducible Conjugate Gradient

Strong scaling results on Tintorrum

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

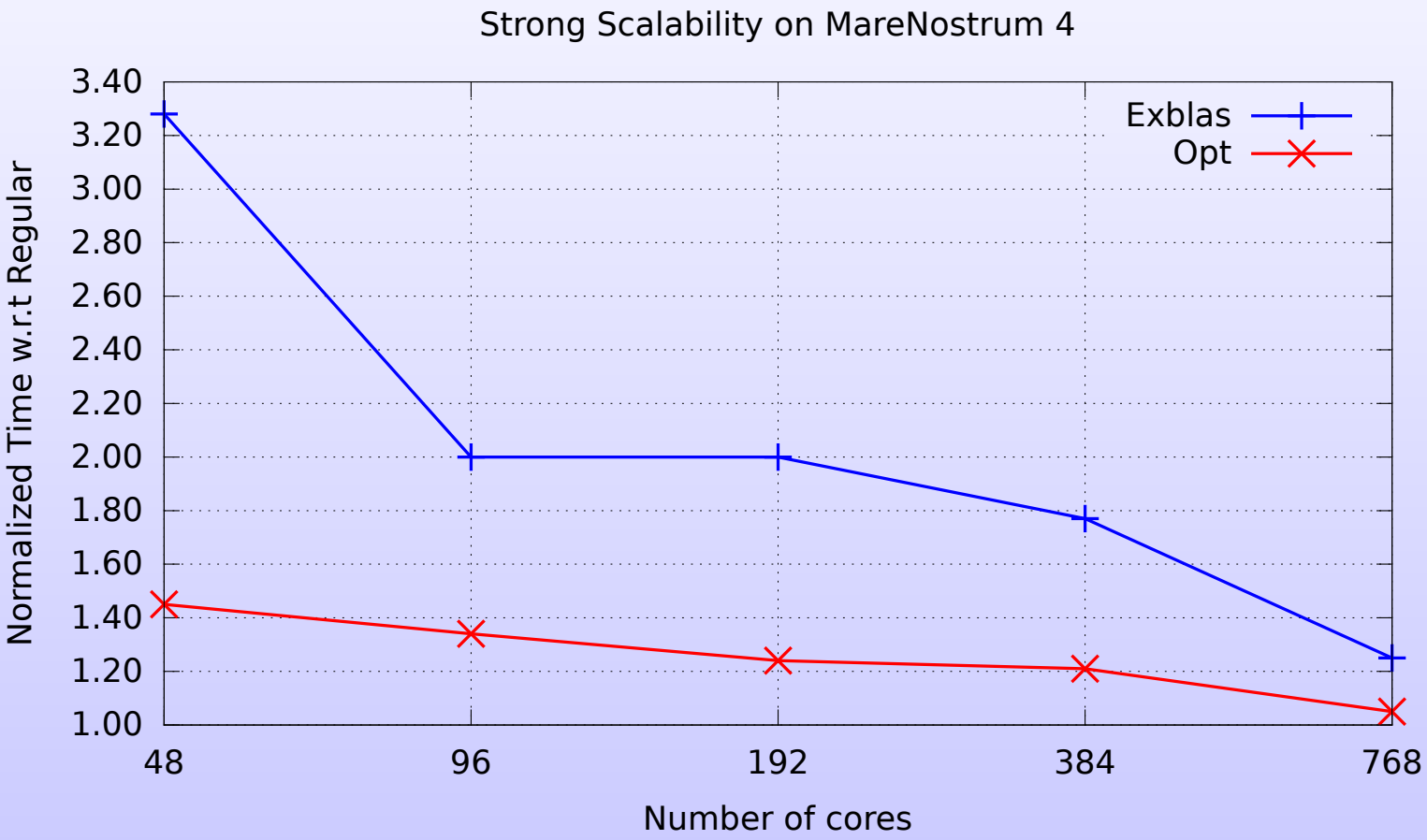


Tintorrum nodes have two 8-core Intel Xeon(R) E5-2630 (Haswell-EP) CPUs @2.4 GHz and 64 GBs of DDR3

Reproducible Conjugate Gradient

Strong scaling results on MareNostrum4

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



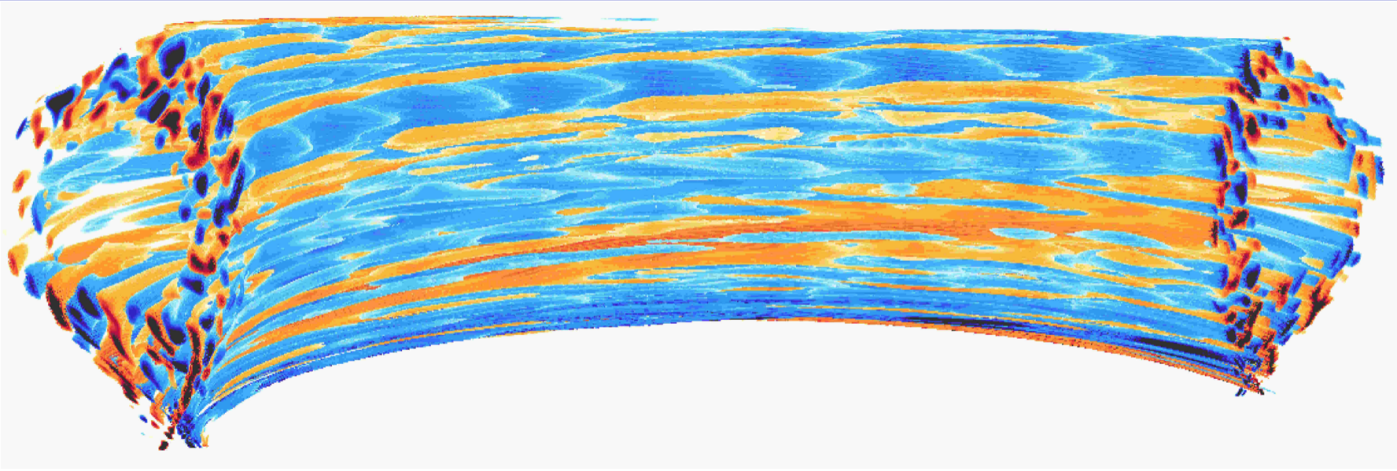
MN4 (BSC) nodes have two 24-core Intel Xeon Platinum 8160 CPUs @2.1 GHz, 96 GBs of DDR3, and connected with Intel Omni-Path

Outline

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Feltor

Full-F **EL**ectromagnetic code in **TOR**oidal geometry

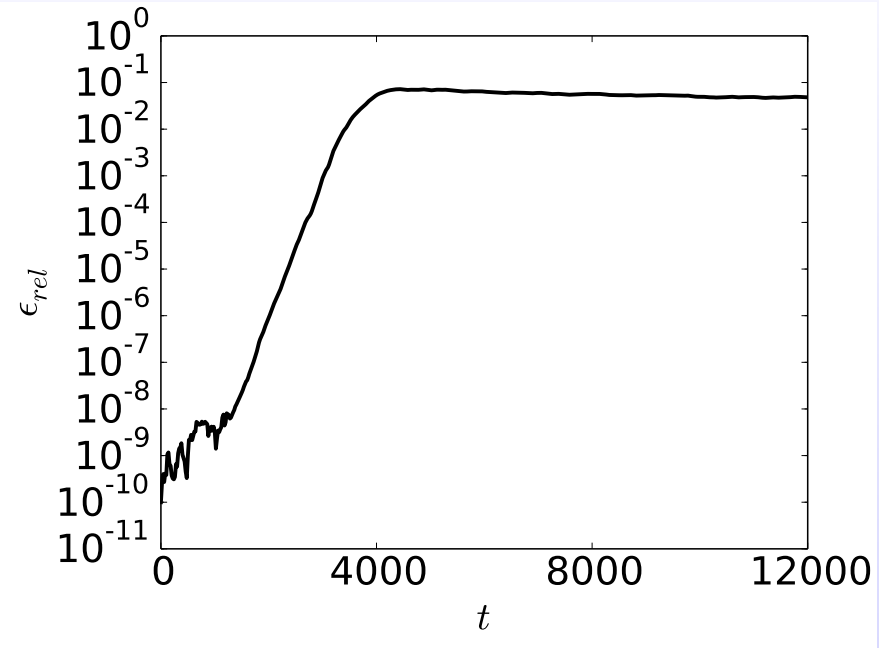


- Both a numerical library and a scientific software package
- 2D and 3D drift- and gyrofluid simulations
- Discontinuous Galerkin methods on structured grids to spatially discretize model equations
- Platform independent code from laptop CPUs to hybrid CPU+GPU distributed memory systems

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Feltor: Accuracy and reproducibility issue

The dimensionless modified full-F Hasegawa-Wakatani model



- $\partial_t n + \{\phi, n\} = \alpha\{\phi - ln(n)\}$
  $\partial_t N + \{\phi - (\nabla\phi)^2/2, N\} = 0$
  $\nabla \cdot (N\nabla\phi) = n - N$

  n – electron density
  N – ion gyro-center density
  ϕ – electric potential

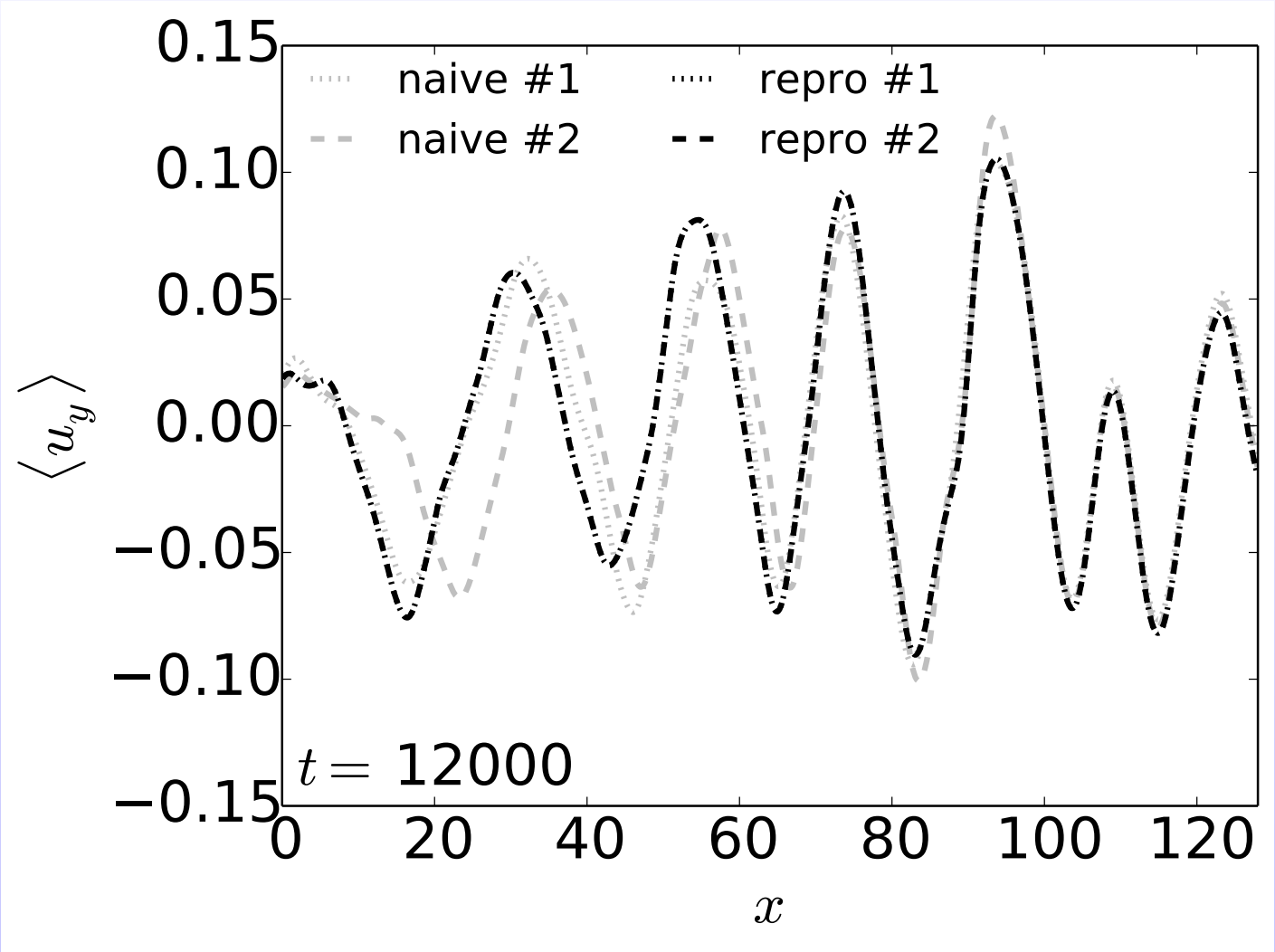
- $e_{rel} = \frac{\|n_1 - n_2\|_2}{\|n_1\|_2}$

Accuracy and reproducibility issue

- Preconditioned Conjugate Gradient (PCG) to invert elliptic equation
- The issue lies in the underlying kernels: `dot`(a,b), `dot`(a,b,c), `axpby`, and `spmv`

Feltor: reproducibility and accuracy

The radial zonal flow structure



Outline

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Conclusion and Future Work

Conclusion

- **ExBLAS** leverages long accumulator and FPEs and often provides correctly-rounded results independently from
  - Data permutation, data assignment, partitioning/blocking
  - Thread scheduling
  - Reduction trees
- Ensured reproducibility of PCG using combined **algorithmic and programming strategies**
- **ExBLAS-based** approach is **generic** and, hence, **robust**, but slow on few nodes. **FPE-only** approach is faster but less generic
- Both approaches show overhead below 30 % at large scale

Future Work

- Detailed error analysis of the FPE-based solution
- Modified FPE-based solution with `AccSum`
- Different preconditioners and pipelined PCGs

Thank you for your attention!

- ExBLAS: `https://github.com/riakymch/exblas`
- Articles: Computer Physics Communications and JCAM SCAN 2018 special issue
- Preprints: `https://arxiv.org/pdf/1807.01971.pdf`
  `https://hal.archives-ouvertes.fr/hal-02391618v1`
- Codes: `https://github.com/feltor-dev/feltor`
  `https://github.com/riakymch/ReproCG`

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257