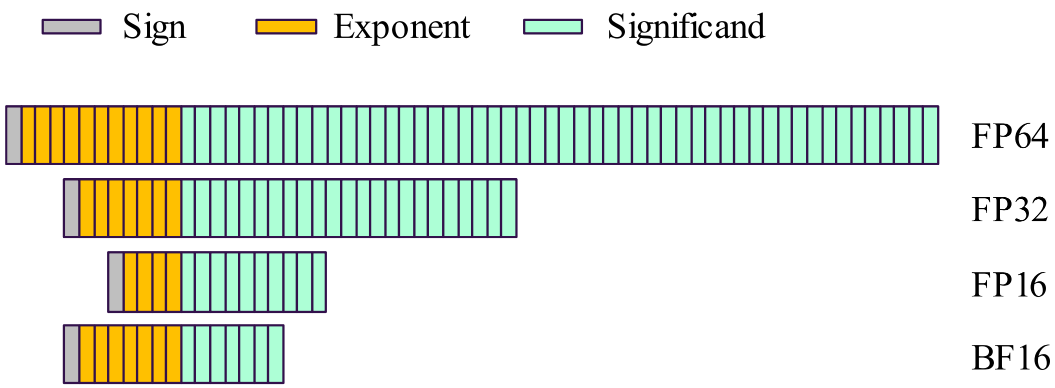
FP16 Acceleration in Structured Multigrid Preconditioner for Real-World Applications

Yi Zong, Peinan Yu, Haopeng Huang, Wei Xue
Dept. Computer Science & Technology
Tsinghua University
Oct 22, 2024 @ MFEM Workshop

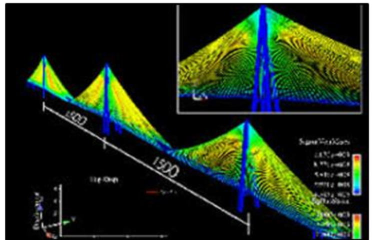
https://doi.org/10.1145/3673038.3673040

Background: Low-Precision Computation

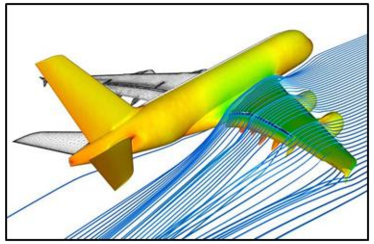
- Half-precision hardware support is now almost ubiquitous.
- We have been witnessing a trend towards lower precision in ML & AI.
- But NOT so prevalent in scientific computation



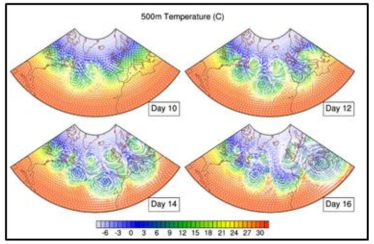
Floating point numbers formats



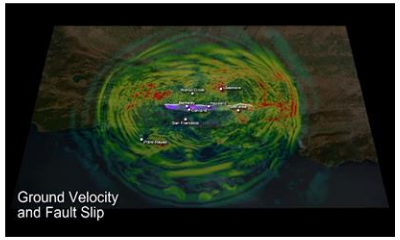
Civil Engineering



Manufacturing



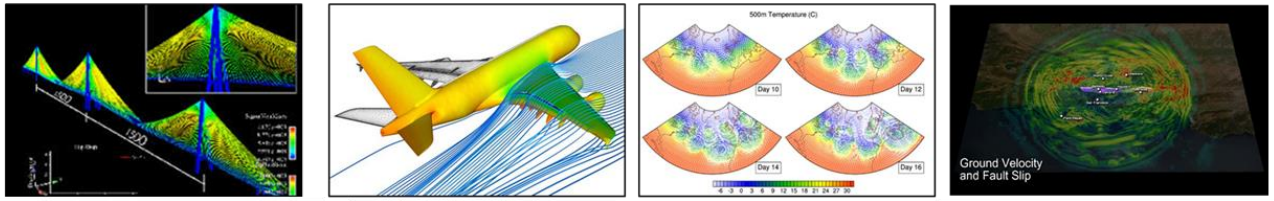
Weather Forecasting



Earthquake Analysis

Background: Low Precision + Linear Solver

- Many scientific simulations rely on numerical solutions to PDE problems
- Essential part: linear solver of $Ax = b$
- Stricter accuracy requirement than AI
- More sensitive to precision



Civil Engineering | Manufacturing | Weather Forecasting | Earthquake Analysis

Application

$$\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) + F(\boldsymbol{x}, t) \quad \cdots\cdots \quad \frac{\partial(\rho \boldsymbol{u})}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u}) = -\nabla \cdot p\boldsymbol{I} + \nabla \cdot \boldsymbol{\tau} + \rho \boldsymbol{g}$$

PDE

$$Ax = b$$

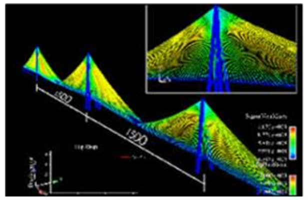
Linear system

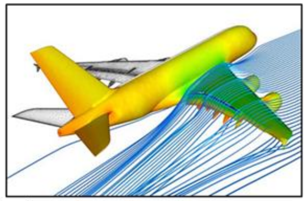
Direct method: $\boldsymbol{O}(\boldsymbol{N}^3)$ **Iterative methods:** $\boldsymbol{O}(\boldsymbol{N}^2)$

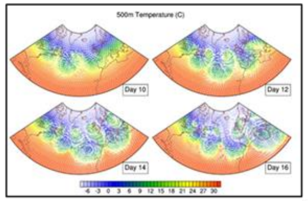
Linear solver

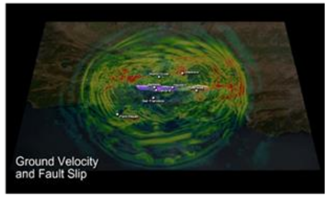
Background: Low Precision + Linear Solver

- Many scientific simulations rely on numerical solutions to PDE problems
- Essential part: linear solver of $Ax = b$
- Stricter accuracy requirement than AI
- More sensitive to precision


Civil Engineering


Manufacturing


Weather Forecasting



Earthquake Analysis

Application

$$\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) + F(\boldsymbol{x}, t) \quad \cdots\cdots \quad \frac{\partial(\rho \boldsymbol{u})}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u}) = -\nabla \cdot p\boldsymbol{I} + \nabla \cdot \boldsymbol{\tau} + \rho \boldsymbol{g}$$

PDE

$$Ax = b$$

Linear system

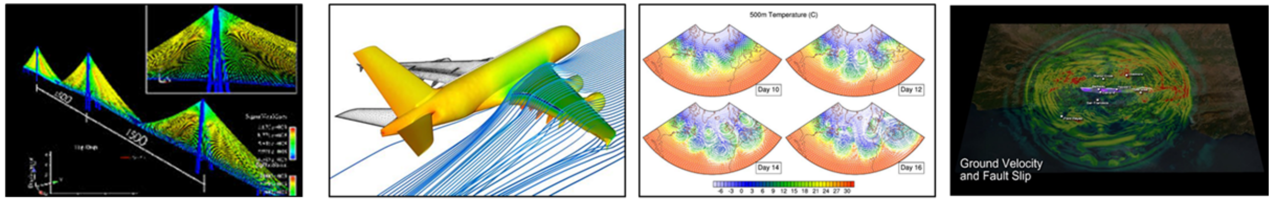
Direct method: $\boldsymbol{O}(\boldsymbol{N}^3)$ Iterative methods: $\boldsymbol{O}(\boldsymbol{N}^2)$

Linear solver

- **Common strategy:**
  - FP64 solver
  - Lower precision preconditioner

Background: Low Precision + Multigrid

- Many scientific simulations rely on numerical solutions to PDE problems
- Essential part: linear solver of $Ax = b$
- Stricter accuracy requirement than AI
- More sensitive to precision



Civil Engineering | Manufacturing | Weather Forecasting | Earthquake Analysis — Application

$$\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) + F(\boldsymbol{x}, t) \quad \cdots\cdots \quad \frac{\partial(\rho \boldsymbol{u})}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u}) = -\nabla \cdot p\boldsymbol{I} + \nabla \cdot \boldsymbol{\tau} + \rho \boldsymbol{g}$$

PDE

- **Common strategy:**
  - FP64 solver
  - Lower precision preconditioner

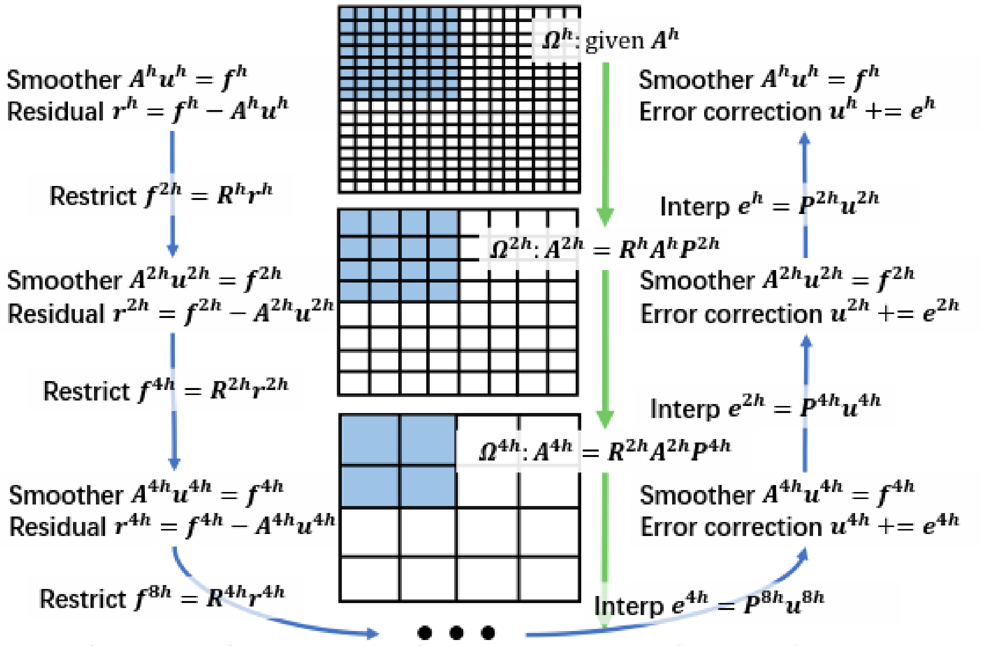
$Ax = b$ — Linear system

Direct method: $\boldsymbol{O}(\boldsymbol{N}^3)$ | Iterative methods: $\boldsymbol{O}(\boldsymbol{N}^2)$ — Linear solver

Multigrid — Preconditioner

Background: Why multigrid (MG) ?

- **Optimal computational complexity** $O(N)$ in solving large-scale sparse linear systems.
- **Widely used** in scientific researches, and industrial software, …
- **Variable smoothers can cover many other "one-level" preconditioners**.
  - Jacobi
  - Gauss-Seidel (GS)
  - Incomplete Lower-Upper (ILU)
  - ……
- **Theoretical foundation**: tolerance of errors introduced by lower-precision [1].
- **Practical benefits**: greater lower-precision speedups than "one-level" preconditioners.
  - A larger proportion of time in workflows

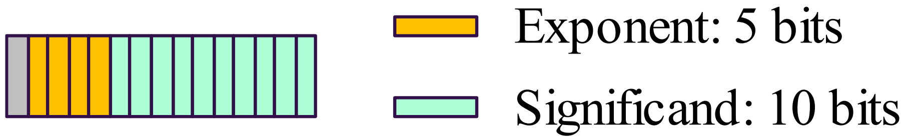


Component	Coarsening	Smoother	Restrict & Interpolate	Residual	Error Correction
Kernel Type	SpGEMM	SpMV/SpTRSV	Stencil/SpMV	SpMV	AXPY

Figure 2: Multigrid overview. V-Cycle is in the solve phase.

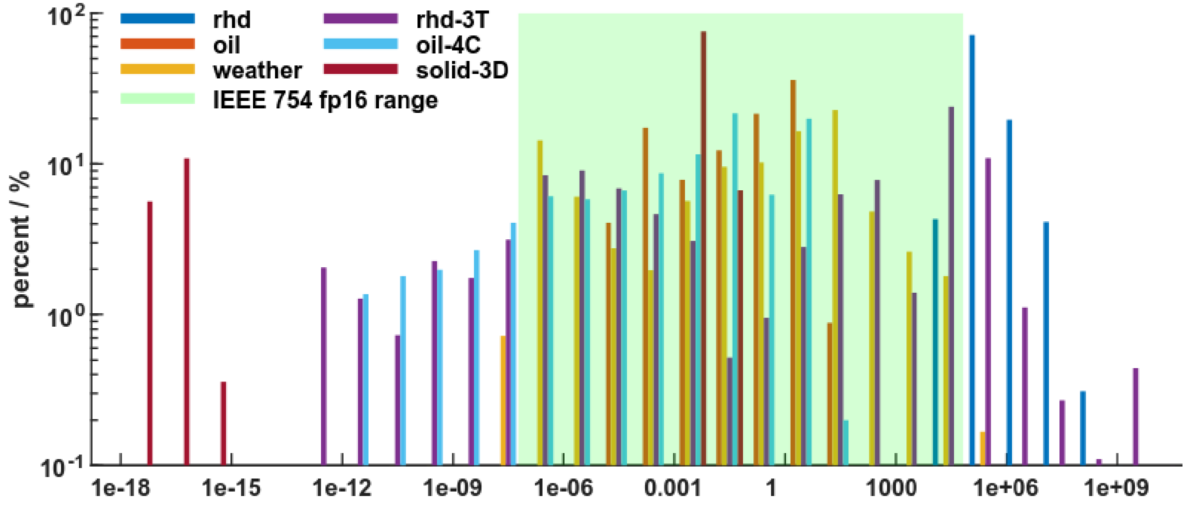
[1] https://doi.org/10.1137/20M1348571

Challenges

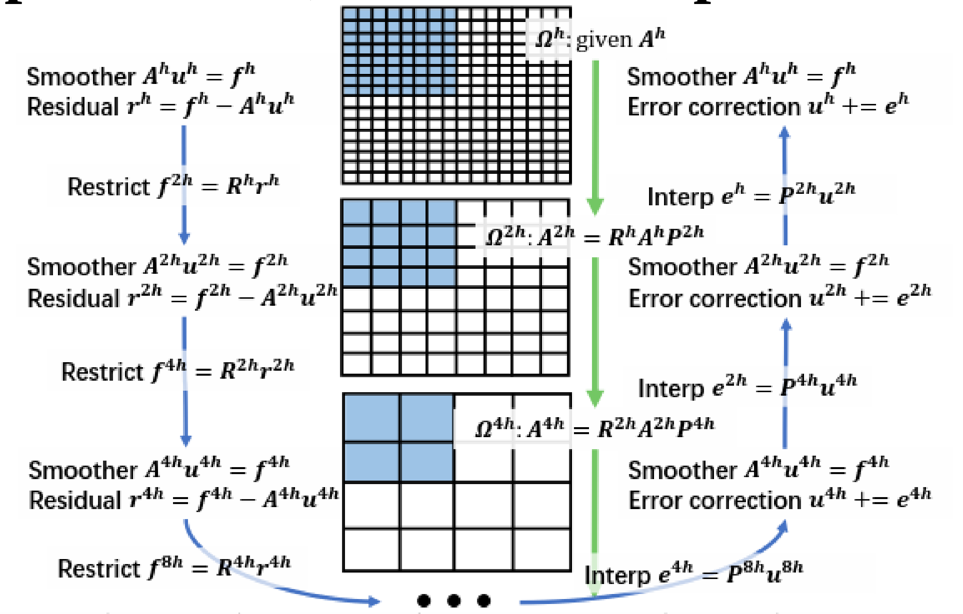


FP16: **narrow range & limited accuracy**

Real world: **Large spans, wide gaps, multi scales** of values



Multigrid: **Multi levels, complicated procedures, variable components**



Component	Coarsening	Smoother	Restrict & Interpolate	Residual	Error Correction
Kernel Type	SpGEMM	SpMV/SpTRSV	Stencil/SpMV	SpMV	AXPY

Related work: Mix-precision in Multigrid

- **Popular AMG libraries (*hypre*, MueLu, AmgX) lack mix-precision support.**
- Practice ahead of theory for a long time
  - **Most of them use only FP32** as the lowest precision in MG preconditioner. They are safe and efficient with similar #iters of mix-FP32/FP64 to those of full-FP64
  - **Utilizing half-precision is scarce**. Ginkgo [6] is a recent three-precision MG, supporting arbitrary precisions (FP64, FP32, FP16) for matrices and vectors on different levels.
  - Lack of guidelines in **how to choose the best configurations**. **Too many changeable options may overwhelm users**.

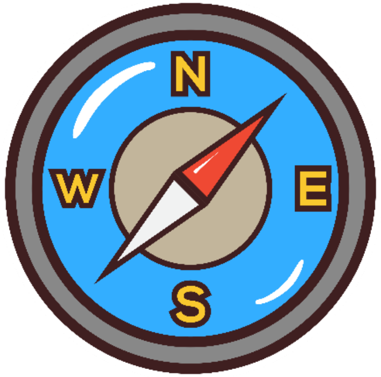
Ref	Type	Scaling?	Precond. Precision	Precond. Speedup	E2E Speedup
[1]	GMG	No	FP32	~2.0x	~1.7x
[2]	AMG	No	FP32	1.1x~1.5x	Unclear
[3]	AMG	No	FP32	Unclear	1.19x
[4]	GMG	No	FP32	1.9x	1.6x
[5]	GMG	No	FP32	2.0x	1.18x
[6]	AMG	Yes	FP16, FP32	Unclear	1.05x~1.35x
Ours	**AMG**	**Yes**	**FP16, FP32**	**2.7x**	**1.9x**

[1] https://doi.org/10.1109/TPDS.2010.61
[2] https://doi.org/10.1016/j.procs.2010.04.020
[3] https://doi.org/10.1049/cp.2014.0185
[4] https://doi.org/10.1007/978-3-642-33134-3_68
[5] https://doi.org/10.1016/j.procs.2016.05.502
[6] https://doi.org/10.1016/j.future.2023.07.024

Related work: Mix-precision in Multigrid

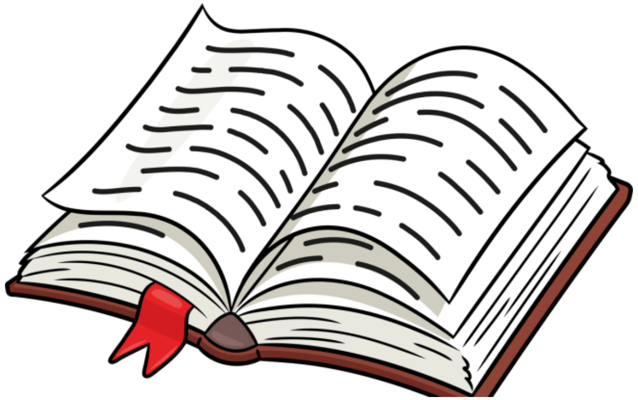
- In our work:

Gains and risks of FP16
Worthwhile combinations



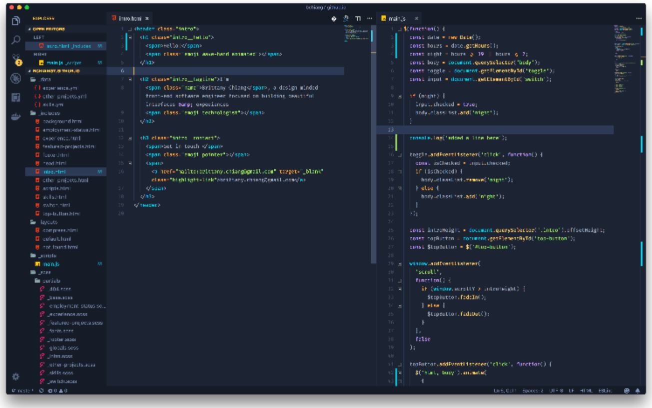
Guidelines

How to adapt setup and solve phases to FP16



Algorithms

How to avoid precision conversion overhead



Implementations

FP16 Utilization Guidelines

- MG's time: $T_{\mathrm{tot}} = T_{\mathrm{setup}} + \#\mathrm{iter} \cdot T_{\mathrm{single}}$
  - T_{setup} is setup time, T_{single} is single-iteration time, #iter is number of iterations.
  - T_{setup} is only slightly affected by mix-precision.
- Lower-precision reduces T_{single}, but may increase #iter.
- Rule: **balance** between **performance** ($T_{\mathrm{single}} \downarrow$) and **convergence** (#iter $\uparrow$)

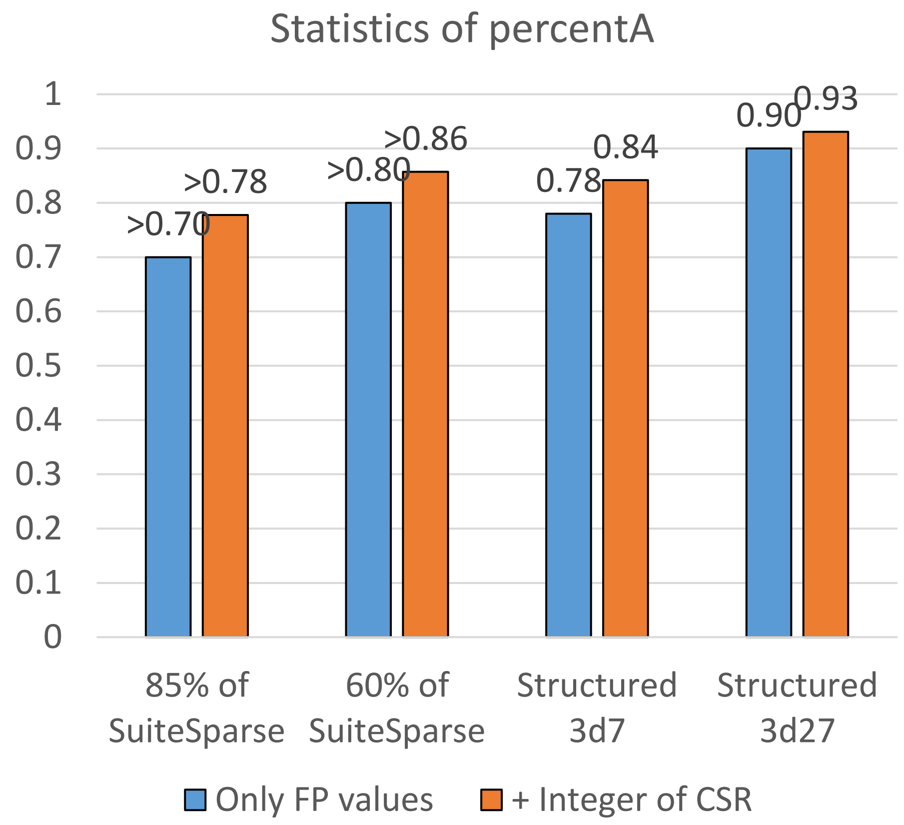
Guideline 1: Eagerly convert matrices to FP16

- **Matrix A is the hotspot of storage.**
- The percent of matrix A in storage in $Ax = b$:

$$\text{percent}_A = \frac{\text{memory}(A)}{\text{memory}(A) + 2 * \text{memory}(x)}$$

- Statistics of matrices indicate **large $\mathbf{percent}_A$.**

- The observation strengthens in MG's multi-level context as the coarser matrices are usually denser [1].
- **Reducing the lower bounds of memory volumes of matrices by FP16 is the top-priority task.**



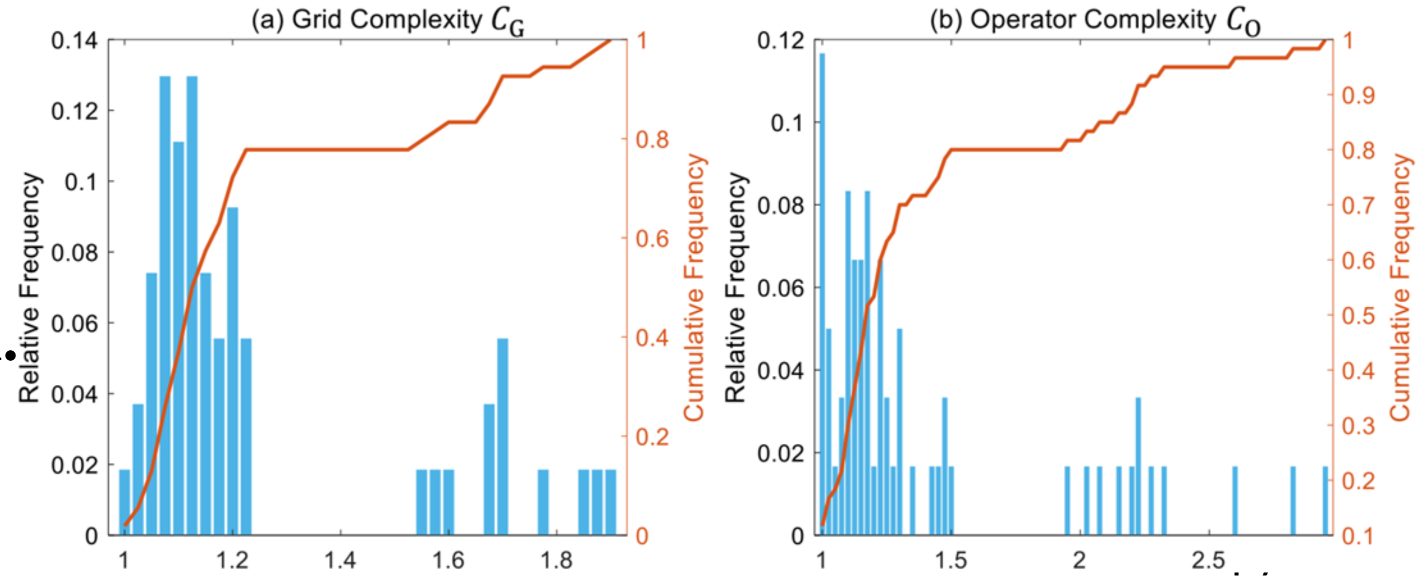
[1] https://doi.org/10.1137/140952570

Guideline 2: Use FP16 from the finest possible level

- Matrices and vectors may have different precisions on different levels.
- 9^n possible combinations for a n-level MG ? **Only limited choices are worthwhile !**
- MG's features: grid complexity C_{G} and operator complexity C_{O}

$$C_{\mathrm{G}} = \frac{\sum_l m_l}{m_0} \qquad C_{\mathrm{O}} = \frac{\sum_l nnz_l}{nnz_0}$$

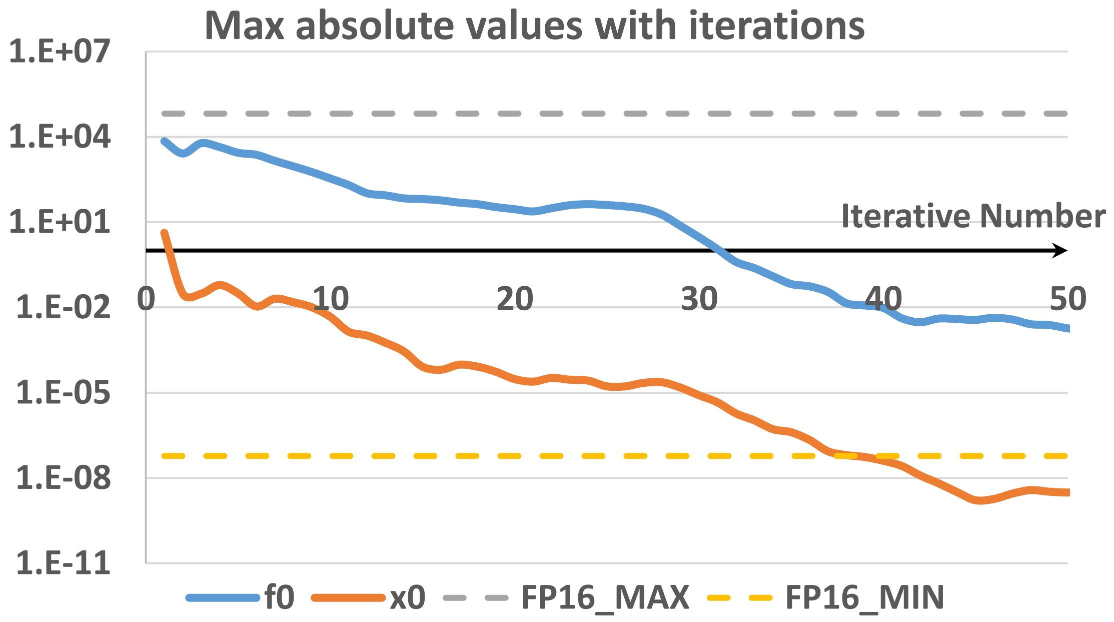
- **Low complexities are common.**
  - $C_{\mathrm{G}} < 1.20$, $C_{\mathrm{O}} < 1.50$ in 80% cases
  - $C_{\mathrm{G}} < 1.15$, $C_{\mathrm{O}} < 1.22$ in 60% cases
- **Most overheads on the finest level.**
- **Common "mistake": use lower precision on coarser levels.**



Complexity statistics for 60 MFEM cases.

Guideline 3: Avoid using FP16 for vectors

- The matrix A is static throughout the solving.
- **The vectors are changing dynamically.**
- Difficult to predict which element may overflow or underflow sometime.



- Safe to keep the precision of vectors $\geq$ FP32, with **limited performance impact**
  - Recalling Guideline 1 that matrix is the hotspot.

Guideline 4: Prefer structured matrix formats that do not use per-element index arrays.

- **Integers in large matrices are difficult to compress.**
  - necessary in unstructured formats, such as CSR, CSC, COO
- Index-free format (e.g., SG-DIA [1]) without integer arrays are more preferred.
  - Can be used in **structured- and semi-structured-specific MGs**, such as SMG, PFMG, SysPFMG, SSAMG from *hypre* (LLNL), RegionMG from *Trilinos* (SNL), and our StructMG, Semi-StructMG, …
- **Upper bound of speedup** can be estimated based on minimal memory access volume.

Format	Bytes per Nonzero			Upper Bound of Speedup	
	FP64	FP32	FP16	FP32/FP16	FP64/FP16
SG-DIA	8	4	2	$\frac{4}{2} = \mathbf{2}$	$\mathbf{4}$
CSR (int32)	$12 + 4\delta$	$8 + 4\delta$	$6 + 4\delta$	$\frac{8 + 4\delta}{6 + 4\delta} < \mathbf{1.3}$	$< \mathbf{2}$
CSR (int64)	$16 + 8\delta$	$12 + 8\delta$	$10 + 8\delta$	$\frac{12 + 8\delta}{10 + 8\delta} < \mathbf{1.2}$	$< \mathbf{1.6}$

$\delta = (\#row + 1)/\#nnz$. $\delta = 15\%$ in average of 2216 matrices in SuiteSparse.

[1] https://doi.org/10.1137/120883153

Guidelines => Algorithms

- The above four guidelines instruct the following **algorithmic designs.**
  - Vectors should be kept in FP32.
  - FP16 should compress the memory volumes of matrices as fine-level as possible.
  - Structured or semi-structured grids should have higher priority to discretize the PDE of interest.

Algorithms notations

- Variables in different colors are in different precisions:

iterative precision usually as **FP64** or **FP32,** is the computation and storage precision of iterative solvers, **determined by users' applications**

computation precision of preconditioners usually as **FP32**

storage precision of preconditioners usually as **FP16**

Setup Algorithm: Setup-then-scale

💡 **Idea: prevent FP16 from interfering with multi-level effect**

- **Essential role: triple matrix products (RAP) that build connections in the hierarchy**
- **A normal setup is completed first** in high precision, followed by scaling (if needed) and truncation.
- A_i **and** Q_i **in high precision are no longer needed** after the setup.
  - Limited additional memory overhead.

Algorithm 1: MG_setup_for_FP16

Input: matrix A

Output: matrices $A_0, A_1, \ldots, A_L$ on hierarchical grids and smoothers $S_0, S_1, \ldots, S_L$, and $Q_0, Q_1, \ldots, Q_L$ if needed

for $i = 0, 1, \ldots, L-1$ **do**
  $A_{i+1} \leftarrow R_i A_i P_{i+1}$ `// Galerkin coarsening`
end
for $i = 0, 1, \ldots, L$ **do**
  if *need to scale* **then** `// truncation after scaling`
    $Q_i \leftarrow \frac{1}{G_i}$ extract_diagonals(A_i);
    $A_i \leftarrow Q_i^{-1/2} A_i Q_i^{-1/2}$;
    $A_i \leftarrow A_i$;
    $Q_i \leftarrow Q_i$;
  else `// direct truncation`
    $A_i \leftarrow A_i$
  end
  $S_i \leftarrow$ smoother_setup(A_i) ; `// setup smoothers`
end

Solve Algorithm: Recover-and-rescale on the fly

- Solve phase consists of the iterative solver and the MG preconditioner.
- **Everything is normal about iterative solvers.**
  - Researches of mix-precision iterative solvers are orthogonal to this study.

- A stationary iterative solver illustrates how an FP16-accelerated preconditioner is employed.
  - Other solvers (CG, GMRES, etc.) are similar.

Algorithm 2: Stationary iterative method.

Input: matrix $\underset{\sim}{A}$, right-hand-side $\underset{\sim}{b}$, initial solution $\underset{\sim}{x}$

Output: approximated solution $\underset{\sim}{x}$

Initialize preconditioner: MG_setup_for_FP16($\underset{\sim}{A}$);

while *not converged* **do**

    $\underset{\sim}{r} \leftarrow \underset{\sim}{b} - \underset{\sim}{Ax}$;

    $\underline{r} \leftarrow \underset{\sim}{r}$; `// truncate residual`

    $\underline{e} \leftarrow$ MG_solve_with_FP16($\underline{r}$) ; `// apply multigrid`

    $\underset{\sim}{e} \leftarrow \underline{e}$; `// recover error`

    $\underset{\sim}{x} \leftarrow \underset{\sim}{x} + \underset{\sim}{e}$;

end

Solve Algorithm: Recover-and-rescale on the fly

💡 Idea : FP16 in storage of matrices (hotspots) to reduce memory access volumes

- **Nothing in iterative precision** inside the MG.
  - Matrices are stored in FP16 and need restoration.
  - Vectors' precision is unchanged, usually as FP32.
- If A_i was scaled, it must be recovered and rescaled
  - **Recover:** $A_i \rightarrow A_i$ (precision promotion)
  - **Rescale:** $A_i \rightarrow Q_i^{1/2} A_i Q_i^{1/2}$ (to original values)
- **On-the-fly:** A_i in FP32 are **NOT** explicitly maintained.

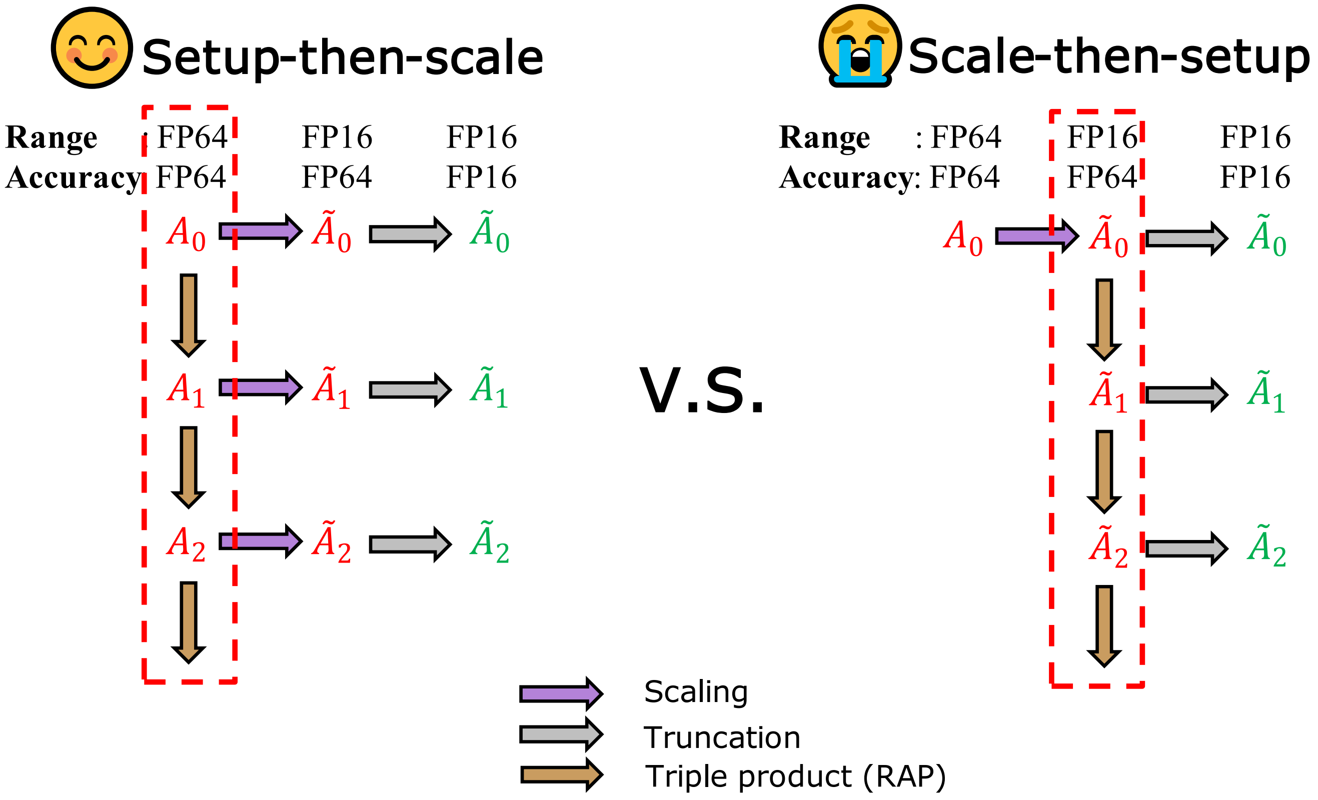
Algorithm 3: MG_solve_with_FP16

Input: right-hand-size b
Output: approximated solution x

Initialize: $f_0 \leftarrow b$;
for $i = 0, 1, ..., L-1$ **do** // forward part of V-Cycle
  for $j = 0, 1, ..., \nu_1 - 1$ **do** // pre-smoothing ν_1 times
    $u_i \leftarrow$ smoother_solve(S_i, f_i, u_i);
  end
  if *scaled in setup* **then** // compute residual
    $r_i \leftarrow f_i - Q_i^{1/2} A_i Q_i^{1/2} u_i$;
  else
    $r_i \leftarrow f_i$ - A_i u_i ;
  end
  if $i < L-1$ **then** // restrict residual to next level
    $f_{i+1} \leftarrow$ Restrict(r_i);
  end
end
for $i = L-1, ..., 1, 0$ **do** // backward parf of V-Cycle
  for $j = 0, 1, ..., \nu_2 - 1$ **do** // post-smoothing ν_2 times
    $u_i \leftarrow$ smoother_solve(S_i^T, f_i, u_i) ;
  end
  if $i > 0$ **then** // interpolate error to next finer grid
    $u_{i-1} \leftarrow u_{i-1}$ + Interpolate(u_i);
  end
end
$x \leftarrow u_0$;

Practical Remarks

- **Additional overhead of Q_i ? Why not scale-then-setup ?**
  - Setup-then-scale has two fold advantages over scale-then-setup.
    - Scaling inside MG without users' involvement
    - Scale-then-setup performs **triple matrix products** (RAP) in FP16 range even if with FP64 precision.
  - In practice, Q_i is cost-efficient compared to more #iter.



Kernel: SIMD to amortize fcvt overhead

- **Mix-precision kernel** introduces additional **overhead of precision conversion (fcvt)**.
- Based on structured-specific MGs that use **AOS** (array of structure) format. Nonzero entries within a row are stored contiguously.

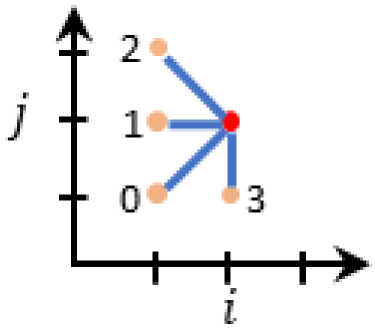
AOS (in FP16)

$A^0_{i,j}$	$A^1_{i,j}$	$A^2_{i,j}$	$A^3_{i,j}$	$A^0_{i,j+1}$	$A^1_{i,j+1}$	$A^2_{i,j+1}$	$A^3_{i,j+1}$	$A^0_{i,j+2}$	$A^1_{i,j+2}$	$A^2_{i,j+2}$	$A^3_{i,j+2}$	$A^0_{i,j+3}$	$A^1_{i,j+3}$	$A^2_{i,j+3}$	$A^3_{i,j+3}$

Scalar instruction for AOS

$$A^0_{i,j} \leftarrow \mathrm{fcvt}\left(\mathrm{ldr}\left(A^0_{i,j}\right)\right)$$
$$A^1_{i,j} \leftarrow \mathrm{fcvt}\left(\mathrm{ldr}\left(A^1_{i,j}\right)\right)$$
$$A^2_{i,j} \leftarrow \mathrm{fcvt}\left(\mathrm{ldr}\left(A^2_{i,j}\right)\right)$$
$$A^3_{i,j} \leftarrow \mathrm{fcvt}\left(\mathrm{ldr}\left(A^3_{i,j}\right)\right)$$

$$\begin{aligned} y[i][j] \leftarrow\ & A^0_{i,j} * x[i-1][j-1] \\ &+ A^1_{i,j} * x[i-1][j\ \ \ ] \\ &+ A^2_{i,j} * x[i-1][j+1] \\ &+ A^3_{i,j} * x[i\ \ \ ][j-1] \end{aligned}$$

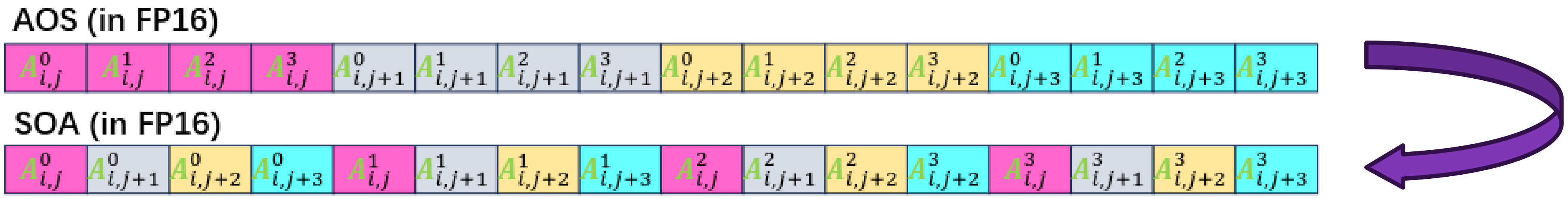


- **AOS is good enough for full-FP32**
  - Only one ldr instruction for each 4-byte entry to prepare for multiplication.
- **For matrices in FP16**, data preparation has **4-times-high arithmetic intensity**
  - One ldr, one fcvt for each 2-byte entry

😭 **Bandwidth efficiency drops !**

Kernel: SIMD to amortize fcvt overhead

- Transforming **from AOS to SOA enables vectorization**
- **Floating-point conversion overhead is amortized.**
- Only one ldr, and two fcvt for every eight 2-byte entries when SIMD length is 128-bit.



Scalar instruction for AOS

$$A^0_{i,j} \leftarrow \mathrm{fcvt}\left(\mathrm{ldr}\left(A^0_{i,j}\right)\right)$$
$$A^1_{i,j} \leftarrow \mathrm{fcvt}\left(\mathrm{ldr}\left(A^1_{i,j}\right)\right)$$
$$A^2_{i,j} \leftarrow \mathrm{fcvt}\left(\mathrm{ldr}\left(A^2_{i,j}\right)\right)$$
$$A^3_{i,j} \leftarrow \mathrm{fcvt}\left(\mathrm{ldr}\left(A^3_{i,j}\right)\right)$$
$$\begin{aligned} y[i][j] \leftarrow\ & A^0_{i,j} * x[i-1][j-1] \\ & + A^1_{i,j} * x[i-1][j] \\ & + A^2_{i,j} * x[i-1][j+1] \\ & + A^3_{i,j} * x[i][j-1] \end{aligned}$$

SIMD for SOA

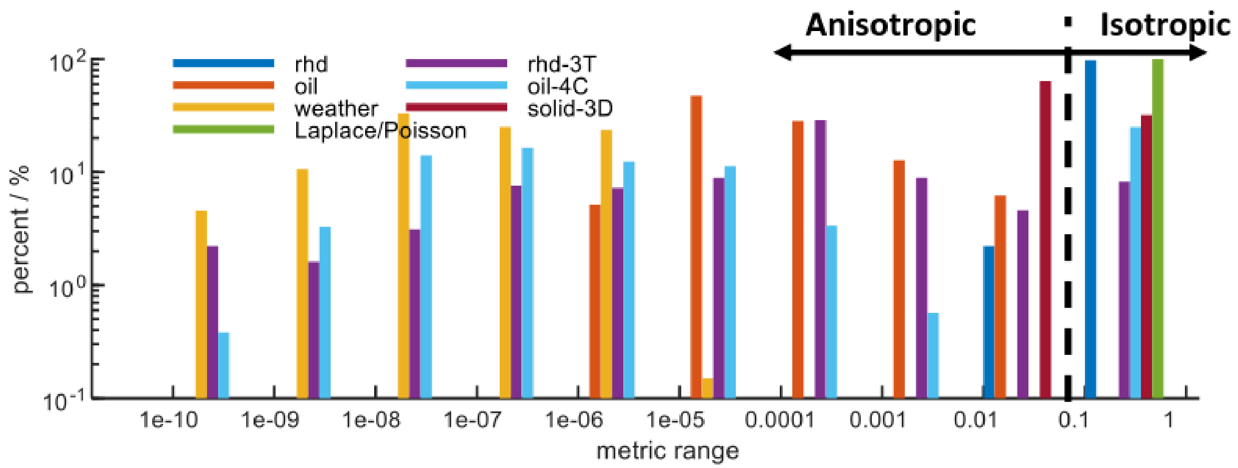
$$A^0_{i,j:(j+3)} \leftarrow \mathrm{fcvt}\left(\mathrm{ldr}\left(A^0_{i,j:(j+3)}\right)\right)$$
$$A^1_{i,j:(j+3)} \leftarrow \mathrm{fcvt}\left(\mathrm{ldr}\left(A^1_{i,j:(j+3)}\right)\right)$$
$$A^2_{i,j:(j+3)} \leftarrow \mathrm{fcvt}\left(\mathrm{ldr}\left(A^2_{i,j:(j+3)}\right)\right)$$
$$A^3_{i,j:(j+3)} \leftarrow \mathrm{fcvt}\left(\mathrm{ldr}\left(A^3_{i,j:(j+3)}\right)\right)$$
$$\begin{aligned} y[i][j:(j+3)] \leftarrow\ & A^0_{i,j:(j+3)} * x[i-1][(j-1):(j+2)] \\ & + A^1_{i,j:(j+3)} * x[i-1][j:(j+3)] \\ & + A^2_{i,j:(j+3)} * x[i-1][(j+1):(j+4)] \\ & + A^3_{i,j:(j+3)} * x[i][(j-1):(j+2)] \end{aligned}$$

Bandwidth efficiency recovers.

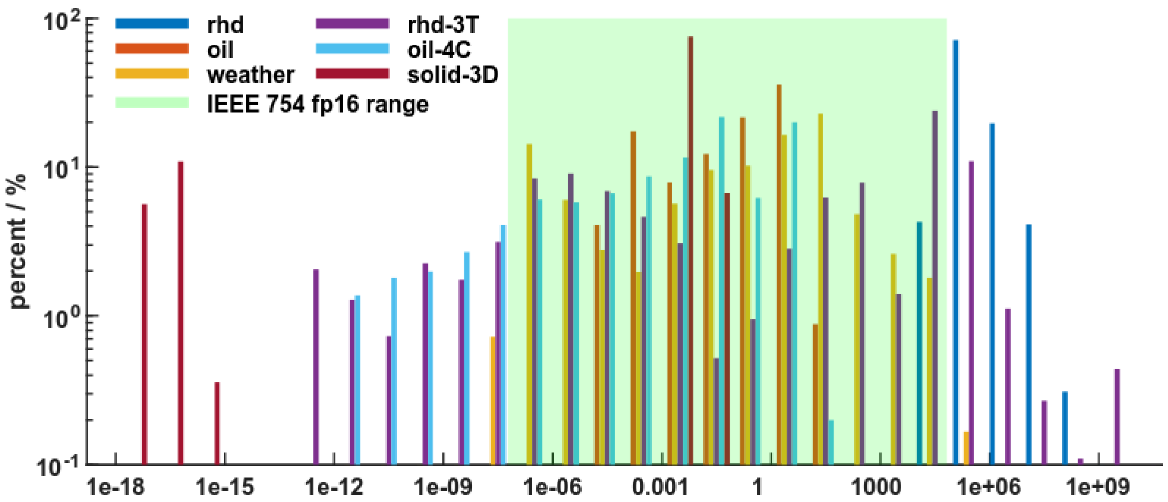
Experiments

- Problems
  - Covering scalar- and vector-type PDE, different and complex numerical features
  - 3 benchmark problems: laplace27, laplace27*10^8, solid-3D
  - 5 real-world problems: rhd, rhd-3D, oil, oil-4C, weather

Problem	Type	Domains
laplace27	Scalar-type PDE	Benchmark problem in HPCG
laplace27*10^8		Coefficients of laplace27 multiplied by 10^8
Rhd		**Radiation hydrodynamics**
Oil		**Petroleum reservoir simulation**. SPE1 and SPE10 benchmarks are combined via OpenCAEPoro.
Weather		**Atmospheric dynamics**, from GRAPES-MESO of 2-km resolution of Chinese region in Dec 2018
rhd-3T	Vector-type PDE	**Radiation hydrodynamics**. 3T: three temperatures (radiation, electron, and ion).
oil-4C		**Petroleum reservoir simulation**. 4C: four components (oil, water, gas, and dissolved gas in oil).
solid-3D		**Linear elasticity in solid mechanics**. 3D: three displacements associated with each element



Anisotropy (i.e., multi-scale property) statistics.



Numerical distributions of nonzero entries.

Problem	Real-world ?	Out-of-FP16 ?	Distance	Anisotropy	Cond. Number	Precision	Solver	C_G	C_O
laplace27	**No**	**No**	**Within**	**None**	**3e+03**	**FP64/FP32/FP16**	CG	1.14	1.14
laplace27*10^8	**No**	**Yes**	**Far**	**None**	**3e+03**	**FP64/FP32/FP16**	CG	1.14	1.14
rhd	**Yes**	**Yes**	**Far**	**Low**	**1e+08**	**FP64/FP32/FP16**	CG	1.14	1.14
oil	**Yes**	**No**	**Within**	**High**	**1e+04**	**FP64/FP32/FP16**	GMRES	1.14	1.14
weather	**Yes**	**Yes**	**Near**	**High**	**1e+05**	**FP32/FP32/FP16**	GMRES	1.31	1.44
rhd-3T	**Yes**	**Yes**	**Far**	**High**	**1e+15**	**FP64/FP32/FP16**	CG	1.14	1.14
oil-4C	**Yes**	**Yes**	**Near**	**High**	**1e+05**	**FP64/FP32/FP16**	GMRES	1.14	1.14
solid-3D	**No**	**Yes**	**Far**	**Low**	**1e+07**	**FP64/FP32/FP16**	CG	1.14	1.26

Experiments

- Solvers
  - Users' applications determine the **iterative precisions**.
  - **StructMG** [1] is used. Other structured-specific MGs can also be used, such as *hypre*'s SMG, PFMG, SysPFMG, SSAMG, …
  - **Only one V-cycle is applied, with one pre- and post-smoothing.**

- Machines
  - ARM and X86 platforms
    - Results are similar on two platforms.
    - ARM with NEON, x86 with AVX2
  - Best results among various MPI/OpenMP ratios reported.

System	ARM	X86
Processor	Kunpeng 920-6426	AMD EPYC-7H12
Frequency	2.60 GHz	2.60~3.30 GHz
Cores per node	128 (64 per socket)	128 (64 per socket)
L1/L2/L3 per core	64 KB/512 KB/1 MB	32 KB/512 KB/4 MB
Stream Triad BW	138 GB/s	100 GB/s
Memory per Node	512 GB DDR4-2933	256 GB DDR4-3200
Max Nodes	64	64
Network	100Gbps InfiniBand	100Gbps InfiniBand
MPI/Compiler	OpenMPI-4.1.4/gcc-9.3.0	Intel-OneAPI-2021.6

[1] https://doi.org/10.1145/3627535.3638482

Results & Analysis

- The results will be presented in a **local-to-global perspective**.

Local → Global

A controlled variables experiment to verify algorithmic effect.

Kernel performance to demonstrate amortizing FP conversion overhead

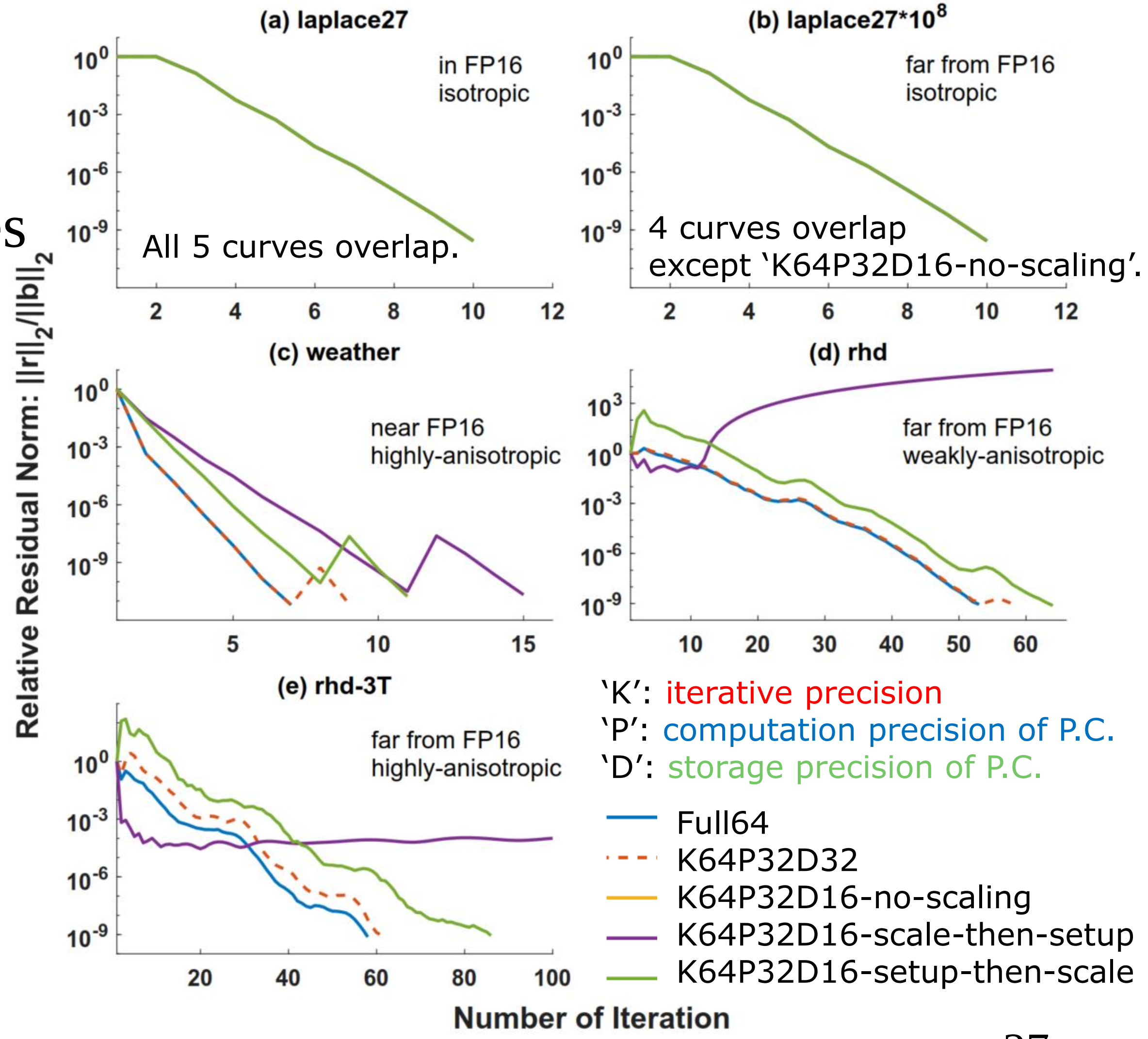
Overall speedups in a single processor

Scalability tests

Results & Analysis: Algorithmic Effect

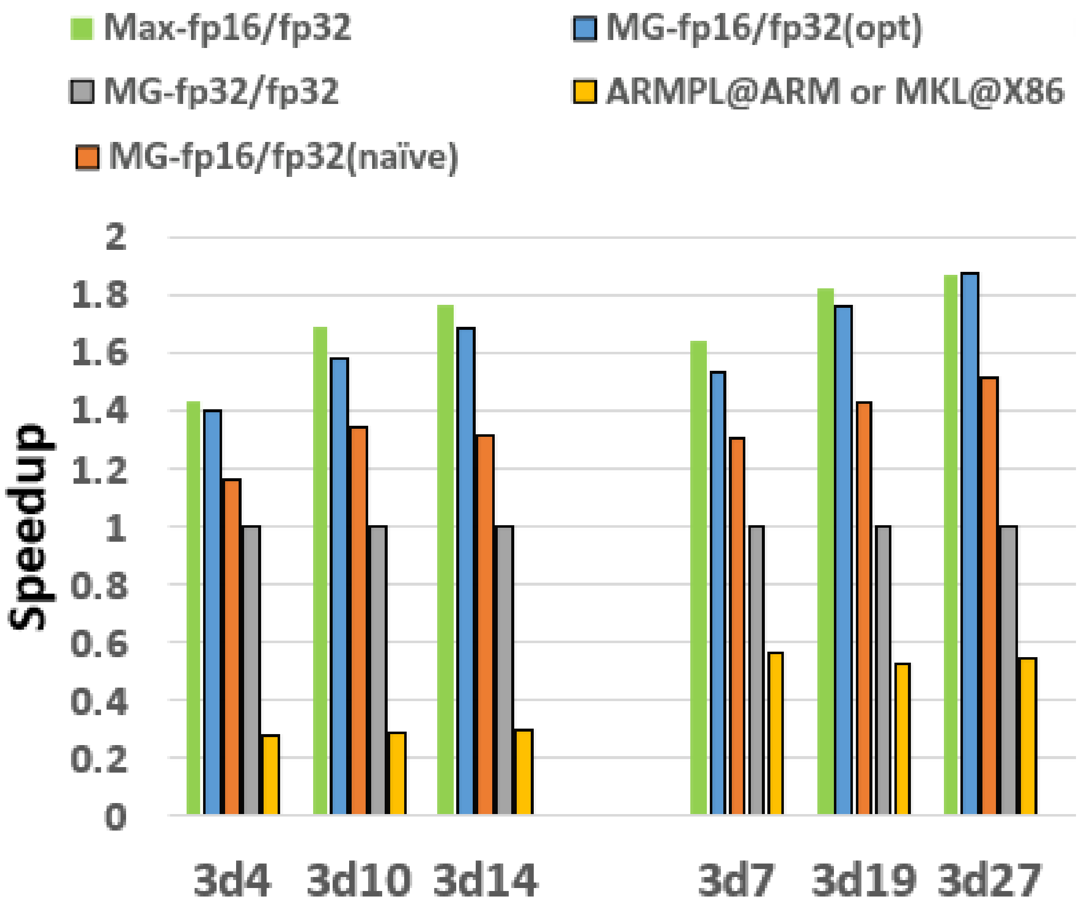
- Descending curves of residual norms in a controlled variable comparison.
- 5 problems feature distinct numerical ranges and distributions.

- Idealized problems cannot distinguish candidates.
- FP16 delays convergence in complicated problems.
- Setup-then-scale is better than scale-then-setup in complicated problems.



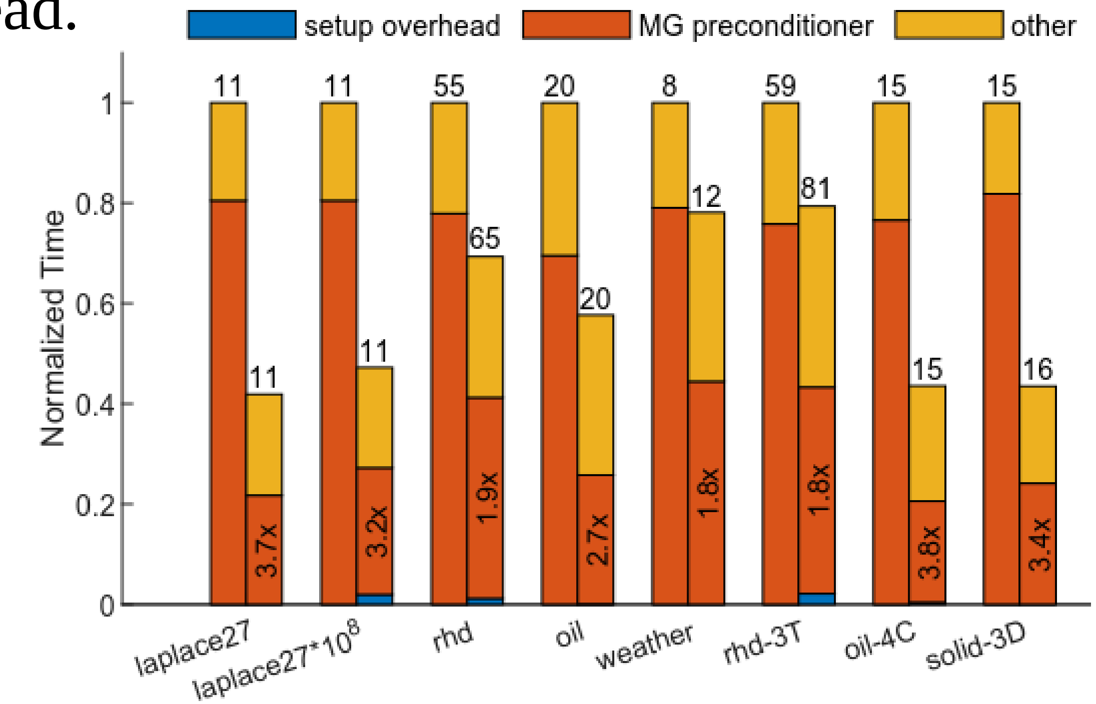
Results & Analysis: Kernel Optimization Effect

- Two essential kernels: SpTRSV and SpMV
- **Baselines: ‘MG-fp32/fp32’**
  - full-FP32 kernels of AOS format, **without** fcvt
  - ~3.5x and ~1.8x faster than ARMPL on ARM
  - ~2.2x and ~1.2x faster than MKL on x86
- **‘Max-fp16/fp32’:** maximum speedups in theory
- **‘MG-fp16/fp32(naive)’:** straightforward extensions of baselines
  - severe bandwidth efficiency degradation
- **‘MG-fp16/fp32(opt)’:** SOA format with SIMD
  - time reduction proportional to memory volume reduction
  - **very close to the theoretical upper bounds**



Results & Analysis: End-to-end Improvement

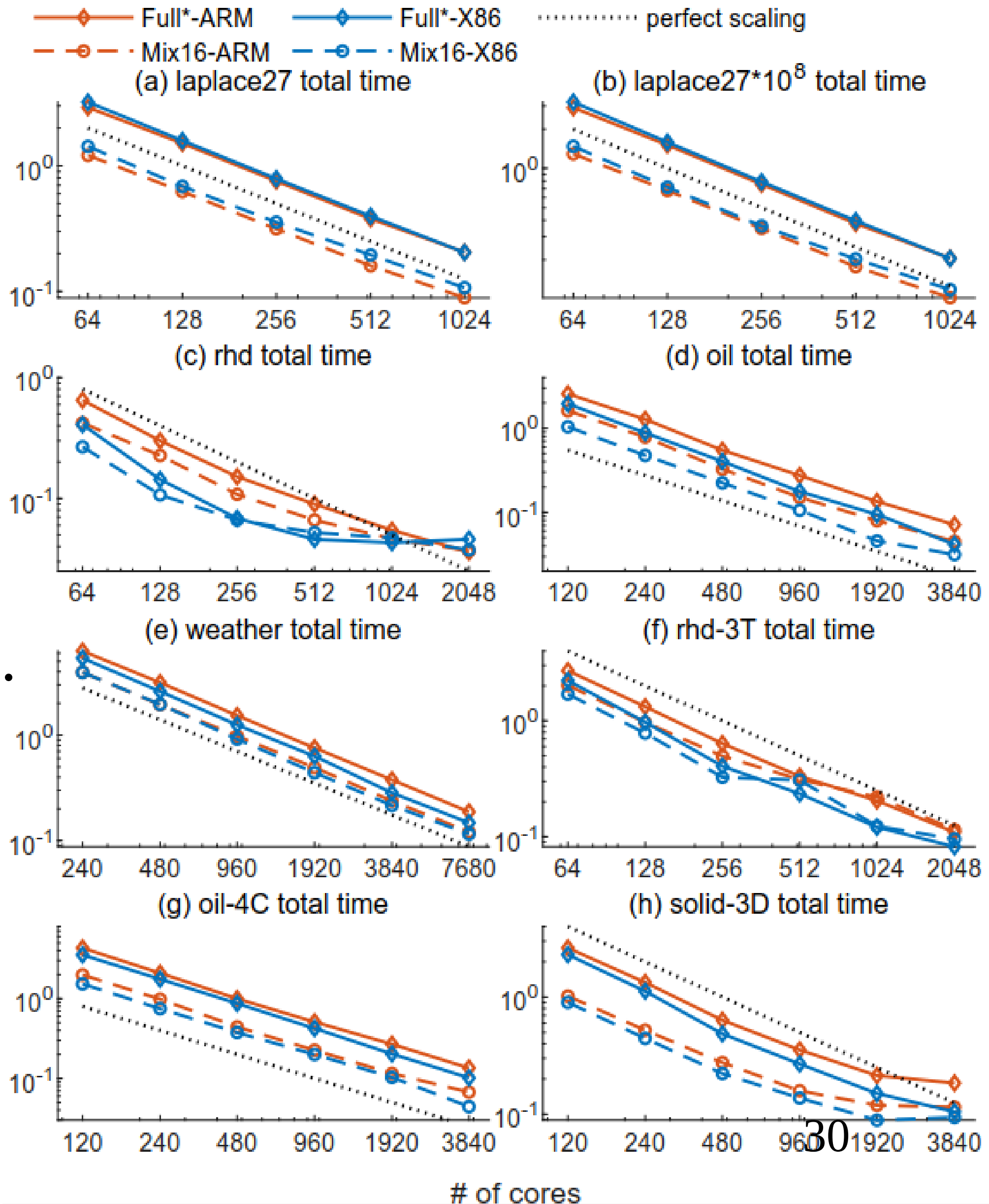
- Setup-then-scale introduces limited setup overhead.
- Speedups are case-dependent.
  - **3.70x** in laplace27: close to the upper-bound (**4.0x**).
  - The additional Q_i reduce speedups.
  - Vector-type PDEs are more favored by FP16.
  - Increases of #iter in rhd, rhd-3T, weather slow down speeds.
- Speedups are 2.4x, 2.2x, 1.7x, 1.7x, 1.9x, 1.8x, 2.3x, 2.4x per iteration for eight problems.



A single processor performance on ARM

Results & Analysis: Scalability Test

- **Mix-precision weakens scalability**
  - FP16 in storage: computation time ↓, comm. proportion ↑.
  - In most cases, leveraging FP16 acceleration at the cost of scalability is **still worthwhile**.
- Parallel efficiencies reach 96%, 89%, 63%, 99%, 98%, 71%, 93%, 62% of the full-iterative-precision counterpart.
- Nearly perfect scalability in medium and large sizes.
  - Given enough workloads per core, speedups are as expected based on reduction of memory volumes.
- Degraded scalability in small sizes.
  - E.g., rhd, rhd-3T, solid-3D
  - **SIMD underutilization** when too few workloads per core.



Discussion (skipped, see the article)

- **1. More numbers of times of V-cycle or smoothing, more significant speedups.**
- **2. BF16 may probably be less suitable than FP16 for linear solvers.**
- **3. Transformation from AOS to SOA extends to GPU.**
- **4. A simple rollback is effective when underflow is severe.**

Conclusion

- Average 2.7x speedups in MG preconditioner and 1.9x in E2E workflow of 8 problems.

- All is about **balance** between **performance** ($T_{\text{single}}\downarrow$) and **convergence** (#iter $\uparrow$).
  - Performance: FP16 in matrices' storage, and recover-and-rescale on the fly
  - Convergence: setup-then-scale avoids damaging multi-level quality

- Guidelines, algorithms, and implementations form the balance together.
  - Guidelines and algorithms also apply to unstructured multigrids with CSR format.
  - Kernel implementations can be ported to *hypre*'s (semi-)structured-specific multigrids, such as SMG, PFMG, SysPFMG, SSAMG, …

Thanks for your listening!

Q & A