# FP16 Acceleration in Structured Multigrid Preconditioner for Real-World Applications

Yi Zong
Tsinghua University
Beijing, China
zong-y21@mails.tsinghua.edu.cn

Peinan Yu
Tsinghua University
Beijing, China

Haopeng Huang
Tsinghua University
Beijing, China

Wei Xue
Tsinghua University
Beijing, China
Qinghai University, Qinghai Provincial Laboratory for Intelligent Computing and Application
Xining, China
xuewei@mail.tsinghua.edu.cn

## ABSTRACT

Half-precision hardware support is now almost ubiquitous. In contrast to its active use in AI, half-precision is less commonly employed in scientific and engineering computing. The valuable proposition of accelerating scientific computing applications using half-precision prompted this study. Focusing on solving sparse linear systems in scientific computing, we explore the technique of utilizing FP16 in multigrid preconditioners. Based on observations of sparse matrix formats, numerical features of scientific applications, and the performance characteristics of multigrid, this study formulates four guidelines for FP16 utilization in multigrid. The proposed algorithm demonstrates how to avoid FP16 overflow through scaling. A setup-then-scale strategy prevents FP16's limited accuracy and narrow range from interfering with the multigrid's numerical properties. Another strategy, recover-and-rescale on the fly, reduces the memory footprint of hotspot kernels. The extra precision-conversion overhead in mix-precision kernels is addressed by the transformation of storage formats and SIMD implementation. Two ablation experiments validate the effectiveness of our algorithm and parallel kernel implementation on ARM and X86 architectures. We further evaluate three idealized and five real-world problems to demonstrate the advantage of utilizing FP16 in a multigrid preconditioner. The average speedups are approximately 2.75x and 1.95x in preconditioner and end-to-end workflow, respectively.

## CCS CONCEPTS

• **Computing methodologies** → **Parallel algorithms**; • **Mathematics of computing** → **Solvers**.



*ICPP '24, August 12–15, 2024, Gotland, Sweden*

ACM ISBN 979-8-4007-1793-2/24/08
https://doi.org/10.1145/3673038.3673040

## KEYWORDS

multigrid, sparse matrix, structured grid, preconditioner

**ACM Reference Format:**
Yi Zong, Peinan Yu, Haopeng Huang, and Wei Xue. 2024. FP16 Acceleration in Structured Multigrid Preconditioner for Real-World Applications. In *The 53rd International Conference on Parallel Processing (ICPP '24), August 12–15, 2024, Gotland, Sweden.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3673038.3673040

## 1 INTRODUCTION

In recent years, there has been a significant increase in hardware support for half-precision floating-point arithmetic [11]. This has been primarily driven by the growing demand for more efficient and high-performance computing. Having witnessed the pivotal shift towards lower precision in the fields of artificial intelligence (AI) and machine learning (ML) [20], it is of particular interest to leverage the benefits of half-precision to achieve significant performance gains in scientific and industrial applications.

These applications, such as numerical weather prediction [21], ocean and groundwater modeling [12], radiation hydrodynamics simulation [34], and solid mechanics [14], primarily rely on numerical simulations, especially numerical solutions to partial differential equation (PDE) problems. The essential part of simulations is solving a linear system $Ax = b$, which usually imposes stricter accuracy requirement than AI, and are more sensitive to precision. Most scientific applications currently use FP64 in the solver, in which a lower-precision preconditioner is a common accelerating technique [1, 10]. Multigrid (MG) is a method of optimal computational complexity $O(N)$ in solving large-scale sparse linear systems [32], and thus widely used as the preconditioner [28].

The opportunity to accelerate multigrid by half-precision lies in its multi-level framework. Its hierarchical grids could be regarded as a series of iterative refinement processes [19], while half-precision iterative refinement itself has been validated in highly ideal experimental conditions, such as HPL-MxP [3]. Unlike other one-level preconditioners, such as incomplete lower-upper (ILU) factorization and symmetric Gauss-Seidel (SymGS), multigrid preconditioners consume more time in solving linear systems. Its dominant role provides a higher upper bound of end-to-end (E2E, i.e., the entire

process of solving the linear system) speedups by half-precision according to Amdahl's Law, as will be seen in our experiments.

However, real-world applications are complicated. The numerical distributions of nonzero entries in six matrices discretized from real-world problems are shown in Figure 1. There are large spans of nonzero magnitudes, and wide gaps among them. The ranges are already beyond the compass of FP16 (IEEE 754 Standard). An even worse problem is that accuracy may be lost near the upper bound of FP16, introducing further convergence issues. BrainFloat16 (BF16) [13] is another half-precision format with the same range as FP32. However, with fewer bits to represent significands, its worse accuracy is worrying to use in scientific computations.

More complicated challenges arise also from the multi-level framework. Matrices and vectors on different levels could be assigned as different precisions. There are $9^n$ (three kinds of precision for matrices multiplied by three kinds for vectors give 9 combinations on a specific level) possible combinations of mix-precision in a $n$-level multigrid, which makes searching for the best combination difficult. Knowing how to find a good-performance combination in theory is of high significance. Meanwhile, the numerical hazards (i.e., overflow and underflow) and accuracy loss of half-precision should be treated more carefully since the connection between two consecutive levels in algebraic multigrids is determined by a triple-matrix product. Predicting the error propagation along a chain of triple-matrix products is difficult. In addition to the above algorithmic issues, implementation matters as the extra precision-conversion overhead may slow down half-precision speed. The mix-precision kernels require instruction-level optimization even if the numerical stability is guaranteed. These issues will be addressed in the following from a top-down perspective.

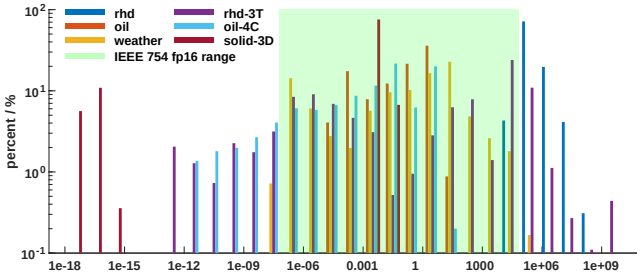


**Figure 1: Numerical distributions of nonzero entries in six real-world matrices. Detailed descriptions refer to section 6.1.**

In this article, we have only investigated the performance of FP16 defined in the IEEE 754 standard. We have opted for FP16 because it provides higher accuracy and is better suited for scientific computing applications, and has more extensive instruction set support than BF16 on X86 and ARM processors. Specifically, this paper makes the following contributions.

- We analyze the potential gains and risks of applying FP16 in multigrids, and summarize four guidelines. The good-performance combinations of precisions are presented.
- We propose an algorithm to adapt multigrid to the use of FP16. The two strategies, setup-then-scale and recover-and-rescale on the fly, avoid the negative influence of FP16's limited range and accuracy loss.
- We present a high-performance parallel implementation of mix-precision kernels to hide the precision-conversion instruction overhead and achieve E2E speedup for ARM and X86 architectures.
- We evaluate three ideal problems and five real-world problems from four application fields. The average speedups of MG preconditioner reach **2.7x** and **2.8x** on ARM and X86, respectively, while the average speedups of the entire workflow are **1.9x** and **2.0x**.

## 2 RELATED WORK

The background of multigrid is first revisited. An overview of multigrid is shown in Figure 2. Based on hierarchical grids, multigrid eliminates high-frequency errors on the finer grid by smoothers, calculates and restricts residuals to the next coarser grid, and computes coarse-grid corrections recursively. Depending on how coarser grids are generated, there are two types of multigrid methods. Geometric multigrid (GMG) rediscretizes the PDE of interest in coarser resolutions [32]. Because rediscretization requires users' explicit involvement with application-specific knowledge, GMG is unavailable in mainstream libraries. A more black-box and user-friendly solution is algebraic multigrid (AMG), which only utilizes information from the assembled finest-level matrix $A^h$ to generate all coarser grids automatically.

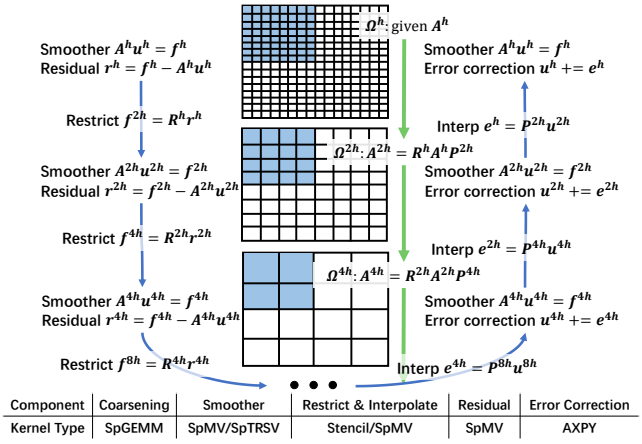


**Figure 2: Multigrid overview. V-Cycle is in the solve phase.**

Multigrid is set up once and solved iteratively. Thus, the total time $T_{\text{tot}}$ of employing a multigrid in iterative solvers is

$$T_{\text{tot}} = T_{\text{setup}} + \#\text{iter} \cdot T_{\text{single}} \quad (1)$$

where $T_{\text{setup}}$ is setup time, $T_{\text{single}}$ is single-iteration solve time, and #iter is number of iterations.

The setup phase is denoted by green arrows in Figure 2. Multigrid constructs a series of coarser grids $\Omega^{2h}, \Omega^{4h}, \ldots$, based on the finest-level grid $\Omega^h$, where the superscripts denote grid spacings of levels. The essential process of setup is the triple-matrix-product in computing the coarser operator (i.e., matrix) $A^{2^k h}$ based on $A^{2^{k-1} h}$ of the finer level.

In the solve phase, denoted by blue arrows, multigrid executes a V-Cycle [32], which starts from the finest level, traverses to the

coarsest, and then reverses. On each level, multigrid invokes a smoother to solve $A^{*h}u^{*h} = f^{*h}$ approximately, computes residual $r^{*h}$ in the downward pass, and computes error correction to update solution $u^{*h}$, and then invokes the smoother again in the upward pass. Restrictions of residuals $r^{*h}$ and interpolations of errors $e^{*h}$ occur in downward and upward pass, respectively.

*Hypre* [15], MueLu [26], and AmgX [22] are popular AMG libraries, but still lack mix-precision support. The practice of mix-precision multigrid is ahead of the theory. Precisions lower than FP64 reduce the memory volumes to accelerate sparse linear solvers that are typically memory-bounded. As listed in Table 1, most of the previous works mainly focused on using FP32, simply by changing the precision of the multigrid preconditioner from FP64 to FP32 in their workflows. GMG is likely to obtain more promising end-to-end speedups because its matrix-free format could be compressed into 50% of its original memory volume of FP64 by FP32, and 25% by FP16. Meanwhile, AMG for unstructured grid problems needs extra integer indices to store the matrix, and the indices usually cannot be compressed. Only the part of floating point data of unstructured AMG could leverage the benefits of lower-precision. Their practical experience showed that using FP32 is safe and efficient for most scenarios since the number of iterations (#iter) of mix-FP32/FP64 in their problems kept close to the original full-FP64 results.

**Table 1: Summary of mix-precision multigrid preconditioner. 'Scale?" means whether its method includes scaling to avoid overflow, and 'N/N' for not needed. 'P.C.' stands for preconditioner.**

| Ref. | Type | Scale? | P.C. precision | P.C. Speedup | E2E Speedup |
|---|---|---|---|---|---|
| [9] | GMG | N/N | FP32 | ~2.0x | ~1.7x |
| [5] | AMG | N/N | FP32 | 1.1x~1.5x | unclear |
| [27] | AMG | N/N | FP32 | unclear | 1.19x |
| [8] | GMG | N/N | FP32 | 1.9x | 1.6x |
| [35] | GMG | N/N | FP32 | 2.0x | 1.18x |
| [33] | AMG | Yes | FP16/FP32 | unclear | 1.05x~1.35x |
| **Ours** | **AMG** | **Yes** | **FP16/FP32** | **2.75x** | **1.95x** |

The explorations of utilizing half-precision in multigrids are less active than FP32, probably due to its unsafe behaviors of underflow, overflow, or accuracy loss. A model problem, isotropic Poisson with homogeneous Dirichlet boundary condition, was tested with FP16 in [24], where both storage and computation were in FP16. However, its methods did not include out-of-range treatment necessary for real-world matrices. A more recent work developed a three-precision algebraic multigrids [33] as part of the Ginkgo library, which enabled arbitrary combinations of precisions (FP64, FP32, and FP16) on different levels. Eight problems from SuiteSparse matrix collection [4] were tested, and E2E speedups of V-cycle mainly were less than 1.2x.

StructMG [40] is an algebraic multigrid preconditioner for structured grid problems and has faster speed and better scalability than the SOTA library *hypre*. The experiments in this article will be based on StructMG, because other mainstream multigrid libraries are incapable of mix-precision configuration. **Our guidelines and algorithms do NOT make assumptions about the background problems and can be applied to other libraries. Furthermore, our implementation is applicable to *hypre*'s SMG, PFMG, and SysPFMG that are designed for structure-grid problems. It is worth mentioning that our guidelines and algorithms are also applicable to unstructured multigrid. But our subsequent analysis will reveal that unstructured multigrid is not suitable for harnessing the mix-precision advantages.**

## 3 FP16 UTILIZATION GUIDELINES

Lower-precision accelerates each single iteration of multigrid at the cost of poorer convergence (i.e., faster $T_{\text{single}}$ but with more #iter in Equation 1). The previous works have shown that the full-FP32 multigrid as a preconditioner in FP64 iterative methods is safe and efficient since the #iter is kept close to its full-FP64 counterpart. Situations become more complicated with the more range-limited FP16. The basic rule of optimization is to make a balance between the decreasing $T_{\text{single}}$ and the probably increasing #iter.

### 3.1 Eagerly convert matrices to FP16.

For performance concern, in solving the linear system $Ax = b$, matrix $A$ is the hotspot of FP16-compressed storage, because its memory access volume is often as several times large as the vectors $b$ and $x$. Given a $m \times m$ matrix $A$, the two vectors $b$ and $x$ require $2m$ elements, and the percentage of the matrix is

$$\text{percent}_A = \frac{nnz(A)}{nnz(A) + 2m} \times 100\% \tag{2}$$

We make a statistic covering the matrices whose $m > 10^4$ in SuiteSparse, and find that 85% of the matrices have $\text{percent}_A > 0.7$, and 60% of the matrices have $\text{percent}_A > 0.8$. If the integer array indices of CSR (compressed sparse rows) format are considered, $\text{percent}_A$ would be more dominant. For structured matrices, $\text{percent}_A$ are 0.78, 0.88, and 0.90 when their nonzero patterns are 3d7, 3d19, and 3d27, respectively.

The above observations still hold in the context of multigrid, and furthermore, the coarser-level matrices $A^{2^k h}(k \geq 1)$ would have more significant percents than that of the finest-level matrix $A^h$. This is because the Galerkin coarsening makes the nonzero patterns of coarser-level matrices expand [6]. Thus, the top priority and most beneficial task is reducing the lower bounds of memory access volumes of matrices by FP16.

### 3.2 Prefer structured matrix formats that do not use per-element index arrays.

Sparse linear solvers are typically memory-bounded programs. The upper bound of lower-precision speedup could be estimated based on the reduction of memory access volume. Since only the floating-point data of an algebraic multigrid could be compressed, structured grid problems are more favored by FP16. In structured-grid-specific multigrids, such as StructMG [40] and *hypre*'s SMG, PFMG, SysPFMG [16], the matrices are stored in structured-grid-diagonal (SG-DIA) format [18] to avoid extra integer indices arrays that are necessary for compressed sparse formats like CSR, CSC (compressed sparse columns), and COO (coordinate format) to represent unstructured grids. We refer to a matrix as a structured matrix if its nonzero pattern fits the SG-DIA format, which happens when the PDE is discretized on a structured grid.

The upper bounds of lower-precision speedup in the preconditioner of SG-DIA and CSR are listed in Table 2. For unstructured grid problems with CSR format, each nonzero entry requires a floating-point number and an integer index in storage. Additionally, the row pointer (also referred to as row map), an integer array in the length of the number of rows plus 1, is amortized to each nonzero entry by $\delta = (m+1)/nnz(A)$ in Table 2. Their upper bounds are remarkably lower than structured grid problems with SG-DIA format, especially when the problem size is too large that int64 is necessary for indices. The E2E speedup would further decrease, which accords with the previous experiments reported in Table 1. Without loss of generality, unstructured grid problems with CSC or COO format can get results similar to CSR ones.

**Table 2: Estimated upper bound of speedup based on minimal memory access volume of matrix. 'P.C.' stands for preconditioner. $\delta = 15\%$ in average for 2216 square matrices in SuiteSparse.**

| | Bytes per Nonzero | | | Upper Bound of P.C. Speedup | | |
|---|---|---|---|---|---|---|
| | FP64 | FP32 | FP16 | FP64/FP32 | FP32/FP16 | FP64/FP16 |
| SG-DIA | 8 | 4 | 2 | 2 | 2 | 4 |
| CSR int32 | $12+4\delta$ | $8+4\delta$ | $6+4\delta$ | $\frac{12+4\delta}{8+4\delta}$ <1.5 | $\frac{8+4\delta}{6+4\delta}$ <1.3 | $\frac{12+4\delta}{6+4\delta}$ <2 |
| CSR int64 | $16+8\delta$ | $12+8\delta$ | $10+8\delta$ | $\frac{16+8\delta}{12+8\delta}$ <1.3 | $\frac{12+8\delta}{10+8\delta}$ <1.2 | $\frac{16+8\delta}{10+8\delta}$ <1.6 |

## 3.3 Use FP16 at the finest possible level.

Matrices and vectors on different levels may have different precisions. The number of possible combinations of precision grows exponentially with the number of levels. However, we will demonstrate that only very limited choices are worthwhile based on the distinct feature of multigrid.

Grid complexity $C_G$ and operator complexity $C_O$ [30] are two metrics to evaluate the computational overhead of a multigrid

$$C_G = \frac{\sum_l n_l}{n_0} \quad \text{and} \quad C_O = \frac{\sum_l Z_l}{Z_0} \tag{3}$$

where $n_l$ and $Z_l$ denote the number of unknowns and nonzero entries, respectively, on level $l$. $l = 0$ corresponds to the finest level.

We have evaluated 10 example problems from MFEM [31] that are suitable for multigrid on 11 kinds of input meshes [1] and collected the $C_G$ and $C_O$ in these 60 cases (one problem on one mesh is one case). These representative cases cover most of the applications using MG as a preconditioner. The statistics are shown in Figure 3. An obvious situation is that most cases have low complexities. The cumulative frequency curves verify that $C_G$ and $C_O$ are lower than 1.2 and 1.5, respectively, in 80% of the cases, and lower than 1.15 and 1.22, respectively, in 60% of the cases.

[1]Specifically, ex1p(Laplace), ex4p(Grad-div), ex5p(Darcy), ex6p(Laplace with AMR), ex8p(DPG for Laplace), ex14p(DG Diffusion), ex26p(Multigrid Preconditioner) are tested with meshes including star.mesh, beam-tri.mesh, escher-p2.mesh, mobius-strip.mesh, square-disc-p3.mesh, and rt-2d-q3.mesh. Ex2p(Linear Elasticity), ex17p(DG Linear Elasticity) and ex21p(AMR for linear elasticity) are tested with meshes including beam-tri.mesh, beam-quad.mesh, beam-tet.mesh, beam-hex.mesh, beam-wedge.mesh, and beam-quad-nurbs.mesh. Refer to https://mfem.org/examples for more detailed description.

Their low complexities result from aggressive coarsening [36] in AMG, and high-dimensional coarsening in StructMG. Lower $C_G$ and $C_O$ benefit the overall performance of multigrid, and thus, aggressive coarsening has been recommended in the best practice guide [6] by the *hypre* team. The outlier ($C_G > 1.5$ and $C_O > 2$) in Figure 3 are the cases whose default settings do not include aggressive coarsening.

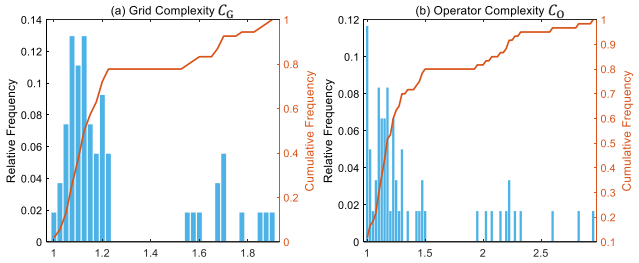


**Figure 3: Complexity statistics for 60 MFEM cases.**

This indicates that the aggregated overhead of all coarser grids is not as significant as half the importance of the finest grid. The dominant role of the finest grid could also be observed in previous works on performance modeling [7]. Therefore, the finer level is accelerated by FP16, the more performance gains will be obtained. FP16 should be utilized to reduce the memory access volumes on levels as fine as possible, which is in contrast to [33] that proposed 'DP-SP-HP'(i.e., the first level's matrix uses double precision, the second level's matrix uses single precision, and the other levels' matrices use half precision). Despite the massive number of possible combinations of precisions in multigrid, only those who apply FP16 on the finer levels are worthwhile from a performance perspective. Based on this observation, multigrid should allow switching the precision from FP16 to higher precisions since a specific level.

## 3.4 Avoid vectors in FP16.

For scientific safety concerns, vectors are not suitable to be stored or computed in FP16 because overflow could happen out of control. In solving a linear system $Ax = b$, the given matrix $A$ is static while the unknown vector $x$ changes dynamically. It is difficult to predict which element of $x$ may overflow sometime. However, the cost of overflow is intolerable. Even if a single element overflows to 'inf', it immediately propagates to 'NaN', crashing the entire program.

The above four guidelines instruct the following algorithmic designs. Vectors should be kept in FP32, while FP16 compresses the memory volumes of matrices as fine-level as possible. Aiming for better leverage of FP16, structured grids should be considered to discretize the PDE of interest.

# 4 ALGORITHMS

This section involves three kinds of precision. Variables of different precision in Algorithm 1, 2, 3 are marked in different colors. The **iterative precision** marked in red with waved underline is the computation and storage precision of iterative solvers, usually as FP64 or FP32, which is determined by users' applications. The **computation precision of preconditioners** is marked in blue

with straight underline, usually as FP32. The **storage precision of preconditioners** is marked in green with dotted underline, usually as FP16.

## 4.1 Setup-then-scale

The strategy in the setup phase is setup-then-scale. As shown in Line 1-3 in Algorithms 1, a normal setup process is completed first in high precision, which includes Galerkin coarsening on all levels.

**Algorithm 1:** MG_setup_for_FP16

**Input:** matrix $\underset{\sim}{A}$
**Output:** matrices $\underset{\sim}{A_0}$, $\underset{\sim}{A_1}$, ..., $\underset{\sim}{A_L}$ on hierarchical grids and smoothers $S_0$, $S_1$, ..., $S_L$, and $\underset{\sim}{Q_0}$, $\underset{\sim}{Q_1}$, ..., $\underset{\sim}{Q_L}$ if needed

1 **for** $i = 0, 1, \ldots, L-1$ **do**
2 $\quad \underset{\sim}{A_{i+1}} \leftarrow \underset{\sim}{R_i}\underset{\sim}{A_i}\underset{\sim}{P_{i+1}}$ // Galerkin coarsening
3 **end**
4 **for** $i = 0, 1, \ldots, L$ **do**
5 $\quad$ **if** *need to scale* **then** // truncation after scaling
6 $\quad\quad \underset{\sim}{Q_i} \leftarrow \frac{1}{G_i}$ extract_diagonals($\underset{\sim}{A_i}$);
7 $\quad\quad \underset{\sim}{A_i} \leftarrow \underset{\sim}{Q_i}^{-1/2}\underset{\sim}{A_i}\underset{\sim}{Q_i}^{-1/2}$;
8 $\quad\quad \underset{\sim}{A_i} \leftarrow \underset{\sim}{A_i}$;
9 $\quad\quad \underset{\sim}{Q_i} \leftarrow \underset{\sim}{Q_i}$;
10 $\quad$ **else** // direct truncation
11 $\quad\quad \underset{\sim}{A_i} \leftarrow \underset{\sim}{A_i}$
12 $\quad$ **end**
13 $\quad S_i \leftarrow$ smoother_setup($\underset{\sim}{A_i}$) ; // setup smoothers
14 **end**

The special treatment to adapt to FP16 is the *need to scale* branch from Line 5 to 9. The purpose of scaling is to transform nonzero values of the matrix $\underset{\sim}{A_i}$ into FP16 range, where $G_i$ is a constant parameter. We have the following theorem to ensure the safety of FP16.

Theorem 4.1. *Truncation to FP16 would not cause unsafe 'inf' if $G_i$ is appropriately chosen.*

Proof. Since scaling operations are independent across different levels, level indices are ommited in the proof. Given the matrix on a specific level $A(a_{ij}) \in \mathbb{R}^{N\times N}$, the diagonal matrix $Q$ in Algorithm 1 is

$$Q = \frac{1}{G}\text{diag}(a_{11}, a_{22}, ..., a_{NN})$$

The element at $i$-row and $j$-column after multiplication of $Q^{-1/2}AQ^{-1/2}$ will be

$$G\frac{a_{ij}}{a_{ii}^{1/2}a_{jj}^{1/2}}$$

Let $S :=$ FP16_MAX denote the upper bound of FP16. The absolute sign of $|*|$ is omitted because representation range of floating-point numbers consists of a positive half and a negative half that are completely symmetric around zero. The following condition needs to be satisfied for all $i, j$ to avoid overflow:

$$G\frac{a_{ij}}{a_{ii}^{1/2}a_{jj}^{1/2}} < S$$

When $a_{ij}$ is large, it requires $G$ to be small. Therefore, the maximal $G$ that avoids overflow is

$$G_{\max} = S\max_{i,j}\left\{\frac{a_{ii}^{1/2}a_{jj}^{1/2}}{a_{ij}}\right\}$$

As long as a $G < G_{\max}$ is chosen, overflow of FP16 is avoided in the truncation after scaling. Since that $Q$ is stored in preconditioner computation precision (usually FP32, but not FP16 in any cases), and will not overflow due to a small $G$. Notice that the square-root operation requires all diagonal entries of the original matrix $A$ are positive, which is naturally included by M-matrix property. It does not make new assumptions because the convergent theory of multigrid itself requires $A$ to be an M-matrix [32]. □

Matrices $\underset{\sim}{A_i}$ are truncated to storage precision in Line 8 and diagonal matrices $\underset{\sim}{Q_i}$ are truncated to computation precision in Line 9. If scaling is unnecessary, direct truncation in Line 11 follows the standard setup process. Based on previous observations, to avoid overflow, whether there is a need to scale or not depends on if there exists $\underset{\sim}{A_i}$'s values greater than FP16_MAX.

Finally, high-precision matrices are used to setup corresponding smoothers. Data in smoothers, such as the factorized lower and upper triangular matrices $\tilde{L}, \tilde{U}$ in ILU, are calculated in iterative precision followed by truncation to storage precision. Notice that high-precision $\underset{\sim}{A_i}$ and $\underset{\sim}{Q_i}$ will no longer be used in multigrid preconditioner after the setup phase.

## 4.2 Recover-and-rescale on the fly during solve

All the optimizations in this article focus on preconditioners, so nothing special is applied to iterative solvers. Due to space limitations, we use a stationary iteration to illustrate how an FP16-accelerated preconditioner is employed. Other iterative algorithms, such as Conjugated Gradient (CG) and Generalized Minimal Residual (GMRES), have the same way of invoking the preconditioner.

As shown in Line 3 in Algorithm 2, residual is calculated by high-precision matrices and vectors in iterative solvers. The preconditioner accepts a lower-precision residual vector as input and returns an error vector of the same precision. Explicit precision transitions are in Line 4 and 6.

**Algorithm 2:** Stationary iterative method.

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

1 Initialize preconditioner: MG_setup_for_FP16($\underset{\sim}{A}$);
2 **while** *not converged* **do**
3 $\quad \underset{\sim}{r} \leftarrow \underset{\sim}{b}$ - $\underset{\sim}{A}\underset{\sim}{x}$ ;
4 $\quad \underline{r} \leftarrow \underset{\sim}{r}$ ; // truncate residual
5 $\quad \underline{e} \leftarrow$ MG_solve_with_FP16($\underline{r}$) ; // apply multigrid
6 $\quad \underset{\sim}{e} \leftarrow \underline{e}$ ; // recover error
7 $\quad \underset{\sim}{x} \leftarrow \underset{\sim}{x} + \underset{\sim}{e}$ ;
8 **end**

There is nothing in iterative precision throughout the V-Cycle of multigrid, as shown in Algorithm 3. The basic idea is that matrices' nonzero entries are stored in FP16 and thus need recovering,

while vectors' values throughout the solve phase remain unchanged as FP32. Therefore, matrices must recover and rescale whenever sparse matrix-vector product (SpMV) and sparse triangular solve (SpTRSV) are performed. The way of rescaling is shown in Line 7 of Algorithm 3. Since computation performs in FP32, the diagonal matrices $Q_i$ are used to rescale values of $A_i$.

Note that the recover-and-rescale is on the fly. Matrices data of FP16 is retrieved, transformed into FP32, and used to calculate only when needed. Data of FP32 is not explicitly maintained; otherwise, the advantage of accessing fewer volumes of memory no longer exists. Rescaling in smoother_solve at Line 4 and 17 is similar if $S_i$ was setup using a scaled $A_i$ previously. SymGS and ILU could be configured as the smoothers of different levels by users, and a direct solver for the coarsest level.

**Algorithm 3:** MG_solve_with_FP16

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

1 Initialize: $f_0 \leftarrow b$;
2 **for** $i = 0, 1, ..., L-1$ **do** // forward part of V-Cycle
3  **for** $j = 0, 1, ..., \nu_1 - 1$ **do** // pre-smoothing $\nu_1$ times
4  $u_i \leftarrow$ smoother_solve($S_i$, $f_i$, $u_i$);
5  **end**
6  **if** *scaled in setup* **then** // compute residual
7  $r_i \leftarrow f_i - Q_i^{1/2} A_i Q_i^{1/2} u_i$;
8  **else**
9  $r_i \leftarrow f_i$ - $A_i u_i$ ;
10  **end**
11  **if** $i < L-1$ **then** // restrict residual to next level
12  $f_{i+1} \leftarrow$ Restrict($r_i$);
13  **end**
14 **end**
15 **for** $i = L-1, ..., 1, 0$ **do** // backward parf of V-Cycle
16  **for** $j = 0, 1, ..., \nu_2 - 1$ **do** // post-smoothing $\nu_2$ times
17  $u_i \leftarrow$ smoother_solve($S_i^T$, $f_i$, $u_i$) ;
18  **end**
19  **if** $i > 0$ **then** // interpolate error to next finer grid
20  $u_{i-1} \leftarrow u_{i-1}$ + Interpolate($u_i$);
21  **end**
22 **end**
23 $x \leftarrow u_0$ ;

## 4.3 Practical Remarks

Further improvement of replacing vector with FP16 is limited for performance concerns. In contrast, overflow risks in vectors' values are potential for scientific accuracy concern, as discussed in Section 3.1 and 3.4.

Our method introduces the additional overhead of accessing $Q_i$ in the solve phase. A natural counterpart would be "scale-then-setup", which first scales the problem matrix $A$, then passes it to multigrid to setup matrices $A_i$ on all levels, and finally truncates them to $A_i$. Though the scale-then-setup strategy is free of $Q_i$ inside V-Cycle, our setup-then-scale in Section 4.1 has two-fold advantages. First, our setup-then-scale does not require users' explicit involvement to scale the original problem matrix and keeps multigrid as a "black-box" tool. Most importantly, it does not interfere with the original setup phase, especially the chain of triple-matrix products. Otherwise, scale-then-setup negatively influences the chain because only floating-point numbers in the FP16 range could be used even if triple-matrix products are calculated with FP64, and even worse, may still incur overflow or underflow. In practice of our various problems, the overhead of diagonal matrices $Q_i$ is cost-efficient compared to the increase of #iter, because $Q_i$ only occupies the same amount of memory as a vector. The advantage of setup-then-scale over scale-then-setup will be verified in the ablation experiments in Section 7.1.

Unlike overflow, which inevitably crashes the program by producing NaN values, underflow is not as catastrophic. However, it can sometimes slow down convergence speed and, in extreme cases, lead to non-convergence. Underflow occurs due to the triple-matrix products during the setup phase, where multiplication causes the absolute values of matrices at coarser levels to become increasingly smaller. A simple but effective method to avoid the adverse effects of underflow is to switch back to higher precision such as FP32. Specifically, the latest version of StructMG [40] introduces a tunable parameter called *shift_levid*. From the *shift_levid* level to the coarsest level, matrices are stored in the computation precision of preconditioners rather than in the storage precision. Switching to higher precision at coarser levels does not introduce significant overhead, because the finer levels are the primary hotspots in multigrid methods, as analysed in Section 3.3.

# 5 INSTRUCTION-LEVEL OPTIMIZATION

Based on the FP16 utilization guidelines and the algorithms, the data of matrices and vectors in multigrid are stored in FP16 and FP32, respectively. SpMV and SpTRSV kernels become mix-precision, which introduces the additional overhead of precision conversion. A representative profiling [37] indicated that SymGS smoother (a specialized form of SpTRSV) accounts for 78% of the entire execution time of the HPCG benchmark, while SPMV has the second-largest contribution of 20%. Therefore, the overhead could not be neglected, as will be shown by the results of Section 7.2, and requires instruction-level optimization.

## 5.1 Vectorization: From AOS to SOA

The implementation is based on structured-grid-specific multigrids, such as StructMG, and *hypre*'s SMG, PFMG, and SysPFMG. The matrices are stored in array-of-structure (AOS) format in these multigrids, as shown in Figure 4. The nonzero entries with the same subscripts (e.g., $(i, j)$) are stored contiguously in the memory. The superscripts, 0~3, of the matrix entries indicate which neighboring position corresponds to this matrix entry. For this example, 0 corresponds to $(i-1, j-1)$, and 1, 2, 3 correspond to $(i-1, j)$, $(i-1, j+1)$, $(i, j-1)$, respectively. Alternatively, the structure-of-array (SOA) format shuffles the entries and stores those with the same superscripts contiguously.

In full-FP32 situations, AOS format is usually good enough to fully utilize the memory bandwidth, which requires only one load (ldr) instruction for each 4-byte entry to prepare for multiplication. However, when the matrix is stored in FP16, AOS requires

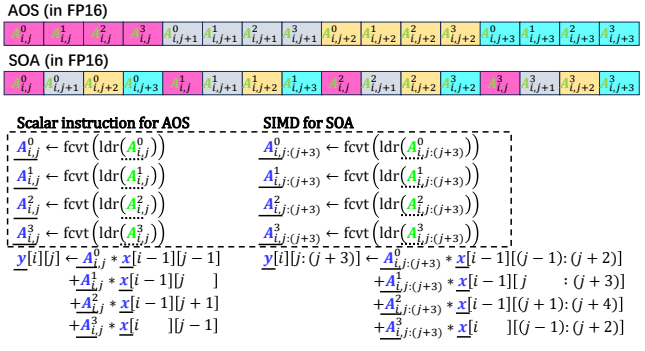


**Figure 4: Illustration of SpMV in AOS and SOA format. Green and blue data are in FP16 and FP32, respectively. The dashed box emphasizes the precision-conversion instructions needed to prepare for FP32 multiplication. SOA is SIMD-friendly. Computations of 4 elements ($j:(j+3)$ written in Matlab style) are vectorized.**

one load, and additionally one floating-point convert (fcvt) instruction for each 2-byte entry. The arithmetic intensity during data preparation is now as 4 times high as the full-FP32 situation. A decrease of bandwidth efficiency[2] will be observed for AOS in the mix-FP16/FP32 situation. Therefore, SOA format is necessary to amortize the precision-conversion overhead via vectorization. As shown in the dashed box in Figure 4, it requires only one ldr, and one fcvt instruction for every four 2-byte entries if the SIMD length is 128-bit. The vectorization of SpMV is straightforward because computations of different elements are independent. SpTRSV requires more sophisticated parallel strategy [39, 40].

## 6 EXPERIMENTAL SETUP

This section describes the problems, solvers settings and machines configurations.

### 6.1 Problems

Our selection of test problems aims to cover different domains and characteristics as extensively as possible. Laplace27 is an idealized benchmark problem in performance evaluation and modeling, such in *hypre*'s reports [17] and HPCG [39]. Laplace27$*10^8$ is an idealized problem that we construct by multiplying the coefficients of laplace27 by $10^8$ to observe the influence of out-of-range of FP16. Rhd and rhd-3T are from radiation hydrodynamics [34]. "3T" means three temperatures (radiation, electron, and ion). Oil and oil-4C are from petroleum reservoir simulation. Settings of SPE1 and SPE10 benchmarks [23] are combined to generate larger cases via OpenCAEPoro [38]. "4C" means four components (oil, water, gas, and dissolved gas in live oil). Weather is from atmospheric dynamics, provided by the dynamic core of GRAPES-MESO [25], the national NWP system of the Chinese Meteorological Administration [2]. The largest case of 637M #dof is a 2km resolution of Chinese regional forecasting in Dec 2018. Solid-3D cases discretized from the weak form of linear elasticity problem in solid mechanics [29], are generated by ourselves. "3D" means three displacements associated with each element. Data of all problems are available online[3].

These problems' characteristics are listed in Table 3. The basic information includes PDE types[4], nonzero patterns ('Pattern' field)[5], total degrees of freedom ('#dof' field), total numbers of nonzero entries ('#nnz' field).

Their numerical features are also considered, including whether the problem is discretized from a real-world application, whether its range is out-of-range of FP16, and the distance from FP16 ('Dist.' field) if it is out. The ranges of different problems and FP16 are displayed in Figure 1. Anisotropy (also referred to as multi-scale property [34]) is an important metric indicating that the characteristics of the linear system vary based on the direction in which they are measured. The more anisotropic a linear system is, the more difficult it is for the solver to converge on it, and the more challenging it is for FP16 to accelerate. The qualitative statements are in the 'Aniso.' field of Table 3, and more detailed visualization can be found in Figure 5. Laplace27 and laplace27$*10^8$ are fully isotropic and have constant coefficients. Rhd-3T is highly anisotropic due to non-smooth coefficients and multi-physics coupling in three-temperature equations. Rhd is relatively isotropic after decoupling from the rhd-3T system. Oil-4C and oil are both highly anisotropic due to inhomogeneous permeability. The strong anisotropy of the weather problem comes from irregular earth topography and nonuniform latitudinal spacing. Solid-3D is relatively isotropic and has homogeneous coefficients. The condition numbers are also included in Table 3. The condition number of weather is evaluated based on a much smaller-size matrix of 9.95M discretized from the same problem, because the original size of 637M is too large to compute the condition number in a reasonable time.

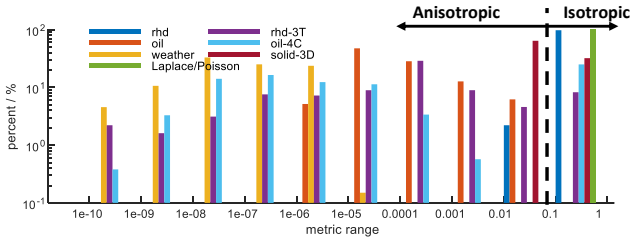


**Figure 5: Statistics of multi-scale metric [34] in six problems.**

### 6.2 Solvers

Iterative Krylov [28] solvers, and the precision corresponding to Section 4 in different problems are in Table 3. Users' applications determine the iterative precisions (in red). The detailed configuration of StructMG as a preconditioner can be found online[3]. All problems have low $C_G$ and $C_O$, which matches our observation and guideline in Section 3.3.

[2]The bandwidth efficiency is the measured bandwidth of the kernel over the bandwidth of the stream benchmark, which can be regarded as the evaluation of architecture efficiency of the kernel. The measured bandwidth refers to the minimal theoretical memory volume to access divided by the measured kernel time.

[3]https://zenodo.org/records/10023590. Data of laplace27 and laplace27$*10^8$ are not included because they can be easily constructed by readers.

[4]Scalar PDE means only one unknown is associated with each grid element, while vector PDE has multiple knowns.

[5]3d15 and 3d19 expand to 3d27 on coarser grids in StructMG and *hypre*'s SMG, PFMG, and SysPFMG.

**Table 3: Test problems characteristics. 'M' for million and 'B' for billion in '#dof' and '#nnz' fields. More detailed statistics of 'Dist.' and 'Aniso.' fields refer to Figure 1 and 5.**

| Problem | Basic Information | | | | Numerical Features | | | | | Solver Information | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PDE | Pattern | #dof | #nnz | Real-world? | Out-of-FP16? | Dist. | Aniso. | Cond. | Precision | Solver | $C_G$ | $C_O$ |
| laplace27 | scalar | 3d27 | 16.8 M | 453 M | No | No | | None | 3e+03 | FP64/ FP32/ FP16 | CG | 1.14 | 1.14 |
| laplace27*$10^8$ | scalar | 3d27 | 16.8 M | 453 M | No | Yes | Far | None | 3e+03 | FP64/ FP32/ FP16 | CG | 1.14 | 1.14 |
| rhd | scalar | 3d7 | 2.10 M | 14.7 M | Yes | Yes | Far | Low | 1e+08 | FP64/ FP32/ FP16 | CG | 1.14 | 1.14 |
| oil | scalar | 3d7 | 31.5 M | 220.2 M | Yes | No | | High | 1e+04 | FP64/ FP32/ FP16 | GMRES | 1.14 | 1.14 |
| weather | scalar | 3d19 | 637 M | 12.1 B | Yes | Yes | Near | High | 1e+05 | FP64/ FP32/ FP16 | GMRES | 1.31 | 1.44 |
| rhd-3T | vector | 3d7 | 6.30 M | 52.4 M | Yes | Yes | Far | High | 1e+15 | FP64/ FP32/ FP16 | CG | 1.14 | 1.14 |
| oil-4C | vector | 3d7 | 31.5 M | 880 M | Yes | Yes | Near | High | 1e+05 | FP64/ FP32/ FP16 | GMRES | 1.14 | 1.14 |
| solid-3D | vector | 3d15 | 11.8 M | 531 M | No | Yes | Far | Low | 1e+07 | FP64/ FP32/ FP16 | CG | 1.14 | 1.26 |

## 6.3 Machines

Experiments are evaluated on ARM and X86 platforms, as shown in Table 4. Similar to [40], the best result at a specific degree of parallelism is reported from tests of various MPI/OpenMP ratios with load-balance process partitions. 1:1 (MPI-only), 1:2, 1:4, 1:8, 1:16, and 1:32 are tested when each NUMA has 32 available cores. 1:1, 1:2, 1:3, 1:5, 1:6, 1:10, 1:15, and 1:30 are tested when only 30 cores are available because ARM's McKernel mechanism reserves two cores for OS to reduce system noise for large-scale tests.

**Table 4: Machines Configurations.**

| 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 |

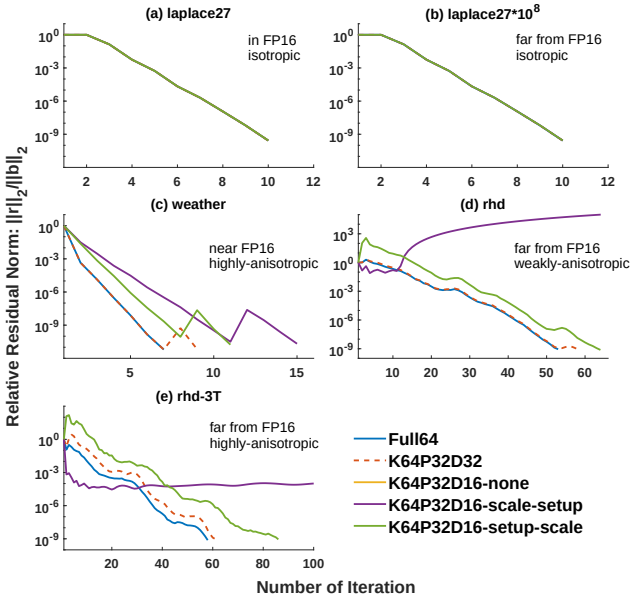


**Figure 6: Descending curves of relative residual norm in five representative problems. 'K', 'P', and 'D' in the legends stand for iterative precision, computation precision of preconditioners, and storage precision of preconditioners in Section 4, respectively. The yellow curve of 'K64P32D16-none' only occurs in (a) because it fails to solve the other problems. The other 4 curves coincide in (b), and all 5 curves coincide in (a). The oscillations in (c) are due to false convergence of GMRES.**

# 7 RESULTS AND ANALYSIS

This section will present the results in a local-to-global perspective. A controlled variables experiment first verifies the algorithmic effect. Kernel performance is then measured to demonstrate the necessity of hiding precision-conversion overhead. The above two effects will be combined to see the overall speedups in a single processor. Finally, strong scalability tests are presented.

## 7.1 Algorithmic Effect

Descending curves of residual norm display how fast a preconditioned solver could converge to the desired solution, as shown in Figure 6. Laplace27, laplace27*$10^8$, weather, rhd, and rhd-3T are analyzed in this ablation experiment, because they have distinct numerical features in different dimensions (i.e., out-of-FP16, distance, and anisotropy) that are labeled in the upper-right corner of sub-figures. A controlled variable comparison could obtain representative conclusions.

Five combinations of precisions and strategies are evaluated. Full64 is the baseline whose curves descend at the highest rate in all problems. K64P32D32 is the FP64 iterative solver preconditioned by an FP32 multigrid, commonly used in previous works. There are three strategies based on K64P32D16 where FP16 is used in storage in the multigrid preconditioner. 'None' means no scaling is applied, and thus would lead to NaN in all problems except laplace27. 'Scale-setup' and 'setup-scale' correspond to the scale-then-setup and setup-then-scale in Section 4.1, respectively. All five combinations perform nearly identically in the idealized benchmark of laplace27, showing completely overlapping residual reduction curves in Figure 6(a). All four combinations, except for 'K64P32D16-none', perform almost identically in the idealized benchmark of

laplace27*$*10^8$, as shown in Figure 6(b). The advantages of setup-then-scale over scale-then-setup are demonstrated in three real-world problems. Even in the weather[6] problem whose values are near the range of FP16, setup-then-scale shows faster-descending curves and results in 11 iterations to converge to $||r||_2/||b||_2$ <1e-10. In comparison, scale-then-setup needs 15 iterations to reach the same convergent threshold. Their differences are further magnified and showcased in rhd and rhd-3T problems, where the values are far from the range of FP16. Scale-then-setup could not converge in these two problems.

## 7.2 Kernel Optimization Effect

Section 5.1 has discussed the extra overhead of precision-conversion instruction, for which the SIMD-friendly SOA format should be used to obtain the expected speedups. The two most essential kernels involving matrices of FP16 in V-cycle are SpMV and SpTRSV, occupying over 80% of multigrid time [37]. We evaluated these two kernels in StructMG on matrices of different patterns. The baselines are full-FP32 kernels of AOS format without precision-conversion overhead (denoted by 'MG-fp32/fp32' in Figure 7). Our baseline SpTRSV and SpMV implementations are ~3.5x and ~1.8x faster than ARM Performance Library (ARMPL, latest version 23.10) on ARM architecture, and ~2.2x and ~1.2x faster than MKL (version 2022.1.0) on X86, respectively. The time of symbolic analysis has been excluded for ARMPL and MKL. More importantly, our mix-precision SpMV and SpTRSV of SOA format with SIMD (denoted by 'MG-fp16/fp32(opt)') could further reduce the kernel times. The time reduction is proportional to the reduction of memory volumes needed to access in the kernel. The higher the ratio of volumes the matrix occupies, the higher the speedup would be. The theoretical maximum reachable speedups (denoted by 'Max-fp16/fp32') are based on the reductions of memory volumes needed to access (i.e., the memory volume of full-FP32 divided by that of mix-precision), which represents the upper-bounds of the mix-precision performance. As shown in Figure 7, our optimized implementations could reach similar speedups with the maximum ones. On the other hand, as we have expected, performance degradation is observed for mix-precision kernels of AOS format (denoted by 'MG-fp16/fp32(naive)'), which are straightforward extensions of the baseline counterparts. The degradation is more pronounced for SpTRSV, which requires sophisticated parallelization. The experiments in Figure 7 utilized all cores within a single NUMA (32 cores on ARM and 64 cores on X86) with multi-threading and speedups are geometrically averaged over problem sizes of $256^3$, $288^3$, $320^3$, $352^3$ and $384^3$.

## 7.3 End-to-end Improvement

Our algorithm aims to minimize the increase of #iter when FP16 is utilized, and the efficient implementation aims to reduce $T_{\text{single}}$ in Equation (1). Combining the two sides leads to an overall high-performance solver preconditioned by FP16-accelerated multigrid.

Figure 8 shows the improvement on a single ARM processor after accelerating the MG preconditioner by FP16 in the entire full-FP64 workflow. All times have been normalized according to the total time of the Full64 solver. The additional overhead of the mix-precision setup is shown in blue columns, which shows that the setup-then-scale strategy introduces only limited overhead during setup. MG preconditioner speedups are indicated in orange columns, and are case-dependent. For the most idealized benchmark problem, laplace27, the speedup **3.70x** approaches the upper-bound of 4.0x shown in Table 2, because the nonzero pattern of 3d27 requires 27 times more memory than a vector. Laplace27*$10^8$ needs

[6]The iterative precision of weather is increased to FP64 in this subsection 7.1 to collect a longer history of descending residual for analysis.

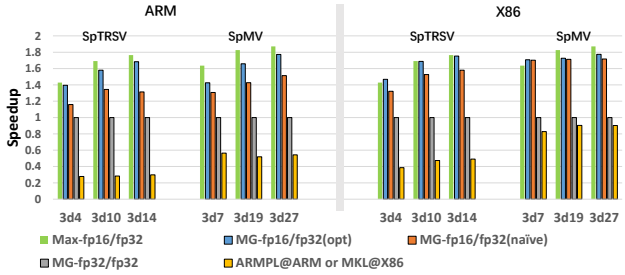


**Figure 7: Ablation experiment of kernel optimization effect. Speedups are over MG-fp32/fp32 (i.e., the best implementation of full-FP32 precision).**

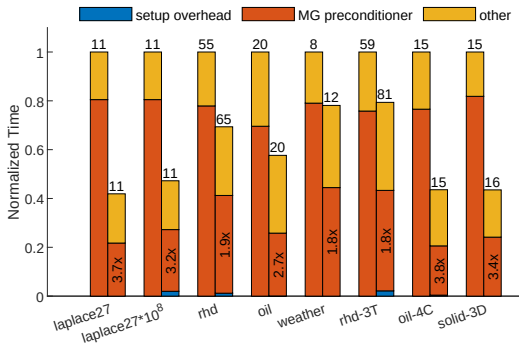


**Figure 8: Performance of solving the linear systems on a single ARM processor. Left column: Full64. Right column: K64P32D16 with the setup-then-scale strategy. The #iters to converge are displayed on the top of columns. Preconditioner speedups are indicated within the orange column.**

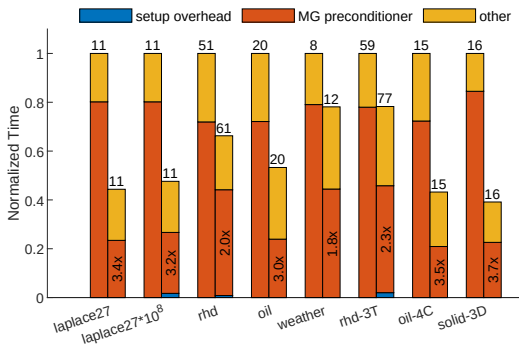


**Figure 9: Performance on a single X86 processor. The legends and meanings are the same as Figure 8.**

an additional diagonal matrix to scale and rescale, thus gaining a slightly lower speedup. Oil's matrix of the 3d7 pattern occupies a smaller proportion of memory than 3d27 and obtains a smaller speedup. Meanwhile, vector PDE problems (especially oil-4C and solid-3D) are more favored by FP16 because each nonzero entry in matrices is a small dense matrix of $r \times r$ where $r$ is the number of variables located at each element. It is worth noting that the increases of #iter in rhd, rhd-3T, and weather slow down their speeds.

The E2E (i.e., the sum of setup overhead, MG preconditioner and other) speedup of the entire workflow could reach **2.39x, 2.21x, 1.73x, 1.74x, 1.92x, 1.78x, 2.32x, 2.45x** for these eight problems on a single ARM processor, respectively. The results are similar on a single X86 processor, as shown in Figure 9.

## 7.4 Scalability Test

Strong scalability is evaluated for the above problems, as shown in Figure 10. The parallel efficiencies on ARM of the mix-precision solver could reach **96%, 89%, 63%, 99%, 98%, 71%, 93%, 62%** of its full-iterative-precision counterpart in terms of total time in these problems, respectively. The scaling behaviors on X86 are similar. Mix-precision solvers could maintain nearly perfect scalability in medium and large-size problems. Their strong scalability will not surpass that of the full-iterative-precision ones because using FP16 in storage accelerates the computation part. In contrast, after optimization, the communication part becomes more dominant in E2E time. Another reason for degraded scalability in small size problems is the underutilization of SIMD when there are too few #dof per core. The extra precision-conversion overhead inhibits the performance in particularly small problems (such as the problems of rhd, rhd-3T and solid-3D). The expected speedups estimated by the reduction of memory volumes could be observed when #dof per core is large enough. In most cases, leveraging FP16 acceleration at the cost of scalability is worthwhile.

## 8 DISCUSSION

In our experiments, both pre-smoothing and post-smoothing are applied once in the V-cycle. Increasing the number of smoothings makes the multigrid method more time-consuming, which results in a more significant E2E speedup when accelerated by half-precision. While some previous studies listed in Table 1 applied smoothings more than once, this article maintains the numbers of smoothing as 1 throughout the experiments. This decision is based on the observation that additional smoothings are generally less efficient in reducing time-to-solution for most problems.

Another half-precision format, BF16, is not discussed in this article. This format will be worth exploring when instruction set support for BF16 becomes more extensive on CPUs. BF16 does not require scaling to avoid overflow because its range is the same as FP32. However, its accuracy is even worse than FP16. In our preliminary evaluation on GPUs, the #iter of using FP16 as the storage precision of preconditioners is always fewer than or equal to that of BF16. An noticeable gap is observed that FP16 and BF16 increase the #iter by 19% and 59%, respectively, compared to the full-FP64 baseline in the rhd problem. Therefore, FP16 appears to be more suitable than BF16 for scientific computations that require strict

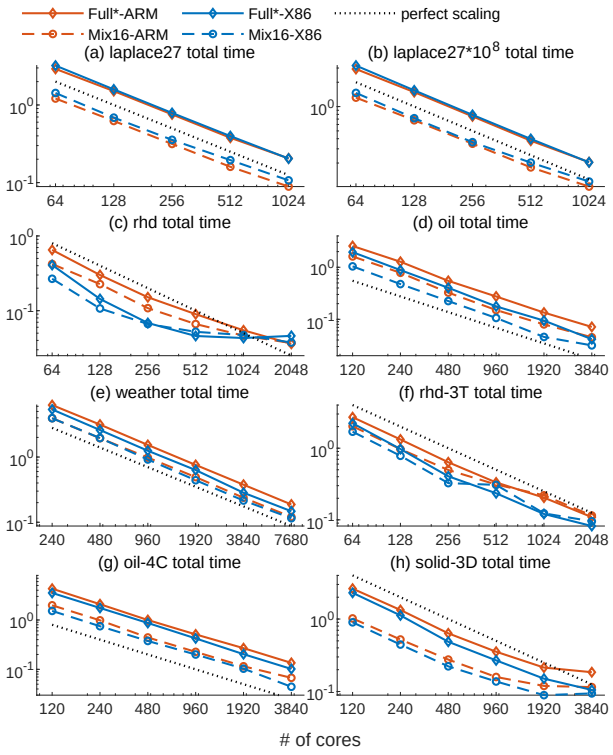


**Figure 10: Results of strong scalability tests. The units of total times are all in seconds. 'Full*' precision stands for the original workflow of the iterative precision, while 'Mix16' means the multigrid preconditioner is configured as the mix-precision in Table 3.**

numerical accuracy. BF16 may need more sophisticated accuracy compensation techniques to leverage its advantage in reducing memory access volume.

The transformation from AOS to SOA format discussed in Section 5.1 extends seamlessly to GPU implementations. Optimizing memory bandwidth efficiency on GPUs necessitates coalesced memory access among threads within a warp. Thus, SOA proves more advantageous even in scenarios utilizing full-FP32 precision.The further precision reduction of FP32 to FP16 for matrices is straightforward based on SOA. Despite new computing hardware innovations on GPUs, represented by tensor cores, they are difficult to leverage significantly for sparse linear algebra kernels that are typically memory-bounded. Similar to CPUs, our GPU implementation has achieved near 100% memory bandwidth efficiency, approaching the theoretical performance limit.

## 9 CONCLUSION

Multigrid and half-precision match with each other. Multigrid provides the numerical tolerance to lower precision due to its multilevel framework [19]. In the meantime, half-precision acceleration obtains much more significant end-to-end speedup because of multigrid's dominance in runtime.

In this article, we investigated the use of FP16 in multigrid preconditioners. Potential gains and risks are analyzed based on the matrix

format and characteristics of multigrid. A complete algorithm is proposed with proof to avoid FP16 overflow. Our setup-then-scale strategy is lightweight in setup and successfully prevents FP16 from interfering with the triple-matrix products. Eight problems (3 idealized, 5 from real-world applications) are evaluated to demonstrate the effectiveness of our algorithms and implementations. Geometric average speedups of **2.7x** and **2.8x** could be obtained in MG preconditioner on ARM and X86, respectively, which contributes average speedups of **1.9x** and **2.0x** in the entire workflow. The maximal speedup **3.8x** in preconditioner has approached the upper-bound of 4.0x and results in a maximal E2E speedup of **2.5x**. Our algorithm makes no assumptions about the background problems and can be applied to other multigrids. The kernel optimization technique can be directly ported to *hypre*'s structured-grid-specific multigrids such as SMG, PFMG, and SysPFMG.

**It is worth mentioning that our guidelines and algorithms are also applicable to unstructured multigrid where matrices are usually stored in CSR format. But it is difficult for unstructured problems to obtain comparable speedups.** The reasons for their inadequate performance stem from two aspects. On one hand, the memory volumes of the extra integer arrays in CSR format could not be reduced by mix-precision storage, as indicated in Table 2. On the other hand, the indirect memory access and inability to leverage vectorization in CSR format further decrease the bandwidth efficiency, as discussed in Section 5.1. These deficiencies significantly offset the programming efforts in utilizing FP16 in unstructured multigrid.

## ACKNOWLEDGMENTS

We would like to thank Xinliang Wang, Qin Wang, Zhaohui Ding, and others for their valuable suggestions. This work is supported by the National Natural Science Foundation of China (NO.U2242210). Wei Xue (Email: xuewei@tsinghua.edu.cn) is the corresponding author of this paper.

## REFERENCES

[1] Ahmad Abdelfattah and et al. 2021. A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. The International Journal of High Performance Computing Applications 35 (Mar 2021).
[2] China Meteorological Administration. 2016. GRAPES Numerical Weather Prediction System. Retrieved July 7, 2023 from https://www.cma.gov.cn/2011xwzx/2011xqxxw/2011xqxyw/202110/t20211030_4079298.html
[3] Innovative Computing Laboratory at University of Tennessee. 2023. HPL-MXP mixed-precision benchmark. Retrieved March 3, 2023 from https://hpl-mxp.org/
[4] Timothy A. Davis and Yifan Hu. 2011. The University of Florida Sparse Matrix Collection. ACM Trans. Math. Softw. 38, 1, Article 1 (Dec 2011), 25 pages.
[5] Maximilian Emans and Albert van der Meer. 2010. Mixed-precision AMG as linear equation solver for definite systems. Procedia Computer Science 1, 1 (2010), 175–183. https://doi.org/10.1016/j.procs.2010.04.020 ICCS 2010.
[6] Robert D. Falgout and Jacob B. Schroder. 2014. Non-Galerkin Coarse Grids for Algebraic Multigrid. SIAM J. Sci. Comput. 36, 3 (Jan 2014), C309–C334.
[7] Hormozd Gahvari and et al. 2012. Modeling the Performance of an Algebraic Multigrid Cycle Using Hybrid MPI/OpenMP. In 2012 41st International Conference on Parallel Processing. 128–137.
[8] S. L. Glimberg and et al. 2013. A Fast GPU-Accelerated Mixed-Precision Strategy for Fully Nonlinear Water Wave Computations. In Numerical Mathematics and Advanced Applications 2011. Berlin, Heidelberg, 645–652.
[9] Dominik Goddeke and Robert Strzodka. 2011. Cyclic Reduction Tridiagonal Solvers on GPUs Applied to Mixed-Precision Multigrid. IEEE Trans. Parallel Distrib. Syst. 22, 1 (jan 2011), 22–32. https://doi.org/10.1109/TPDS.2010.61
[10] Nicholas J. Higham and Theo Mary. 2022. Mixed precision algorithms in numerical linear algebra. Acta Numerica 31 (2022), 347–414.
[11] Nhut-Minh Ho and et al. 2017. Exploiting half precision arithmetic in Nvidia GPUs. In 2017 IEEE High Performance Extreme Computing Conference (HPEC).
[12] X. Huang and et al. 2016. P-CSI v1.0, an accelerated barotropic solver for the high-resolution ocean model component in the Community Earth System Model v2.0. Geoscientific Model Development 9, 11 (2016), 4209–4225.
[13] Intel. 2018. BFLOAT16 - hardware numerics definition. Retrieved Nov 30, 2023 from https://www.intel.com/content/dam/develop/external/us/en/documents/bf16-hardware-numerics-definition-white-paper.pdf
[14] Carlo Janna, Andrea Comerlati, and Giuseppe Gambolati. 2009. A Comparison of Projective and Direct Solvers for Finite Elements in Elastostatics. Adv. Eng. Softw. 40, 8 (aug 2009), 675–685. https://doi.org/10.1016/j.advengsoft.2008.11.010
[15] Lawrence Livermore National Lab. 2023. Documentation for hypre. Retrieved March 3, 2023 from https://hypre.readthedocs.io/en/latest
[16] Lawrence Livermore National Lab. 2023. Structured multigrid in HYPRE. Retrieved March 3, 2023 from https://hypre.readthedocs.io/en/latest/solvers-smg-pfmg.html
[17] Ruipeng Li and Ulrike Meier Yang. 2021. Performance Evaluation of hypre Solvers. (Feb 2021). https://doi.org/10.2172/1764323
[18] Daniel Lowell and et al. 2013. Stencil-Aware GPU Optimization of Iterative Solvers. SIAM Journal on Scientific Computing 35, 5 (2013), S209–S228.
[19] Stephen F. McCormick, Joseph Benzaken, and Rasmus Tamstorf. 2020. Algebraic error analysis for mixed-precision multigrid solvers. SIAM J. Sci. Comput. 43 (2020), S392–S419.
[20] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed Precision Training. arXiv:1710.03740 [cs.AI]
[21] Eike H. Müller and et al. 2014. Massively parallel solvers for elliptic partial differential equations in numerical weather and climate prediction. Quarterly Journal of the Royal Meteorological Society 140, 685 (2014), 2608–2624.
[22] M. Naumov and et al. 2015. AmgX: A Library for GPU Accelerated Algebraic Multigrid and Preconditioned Iterative Methods. SIAM Journal on Scientific Computing 37, 5 (2015), S602–S626.
[23] Society of Petroleum Engineers. 2023. SPE Comparative Solution Project. Retrieved March 3, 2023 from https://www.spe.org/web/csp/datasets/set02.htm
[24] Kyaw Linn Oo and Andreas Vogel. 2020. Accelerating Geometric Multigrid Preconditioning with Half-Precision Arithmetic on GPUs. arXiv:2007.07539 [cs.MS]
[25] China Meteorological News Press. 2014. An Introduction of GRAPES. Retrieved July 7, 2023 from https://www.cma.gov.cn/en/NewsReleases/MetInstruments/201403/t20140327_241784.html
[26] Trilinos project. 2023. MueLu. Retrieved Nov 26, 2023 from https://trilinos.github.io/muelu.html
[27] Christian Richter and et al. 2014. GPU-accelerated mixed precision algebraic multigrid preconditioners for discrete elliptic field problems. In 9th IET International Conference on Computation in Electromagnetics. 1–2.
[28] Yousef Saad. 2003. Iterative Methods for Sparse Linear Systems (second ed.). Society for Industrial and Applied Mathematics.
[29] Martin H. Sadd. 2005. Elasticity: Theory, Applications, and Numerics. Academic Press. https://doi.org/10.1016/B978-0-12-605811-6.X5000-3
[30] K. Stuben. 2000. Algebraic Multigrid (AMG) : An Introduction With Applications.
[31] MFEM team. 2023. MFEM examples. Retrieved Nov 26, 2023 from https://mfem.org/examples
[32] Ulrich Trottenberg and et al. 2001. Multigrid. Academic Press, San Diego, California, USA.
[33] Yu-Hsiang Mike Tsai and et al. 2023. Three-precision algebraic multigrid on GPUs. Future Generations Computer Systems 149 (12 2023).
[34] Xiaowen Xu and et al. 2017. Algebraic interface-based coarsening AMG preconditioner for multi-scale sparse matrices with applications to radiation hydrodynamics computation. Numerical Linear Algebra with Applications 24, 2 (2017), e2078. https://doi.org/10.1002/nla.2078
[35] Takateru Yamagishi and et al. 2016. GPU Acceleration of a Non-Hydrostatic Ocean Model with a Multigrid Poisson/Helmholtz Solver. Procedia Comput. Sci. 80, C (jun 2016), 1658–1669. https://doi.org/10.1016/j.procs.2016.05.502
[36] Ulrike Meier Yang. 2010. On long-range interpolation operators for aggressive coarsening. Numerical Linear Algebra with Applications 17, 2-3 (2010), 453–472.
[37] Xiaojian Yang and et al. 2023. Optimizing Multi-Grid Computation and Parallelization on Multi-Cores. In Proceedings of the 37th International Conference on Supercomputing (ICS '23). 227–239. https://doi.org/10.1145/3577193.3593726
[38] Chensong Zhang and et al. 2023. OpenCAEPoro. Retrieved March 3, 2023 from https://github.com/OpenCAEPlus/OpenCAEPoro/tree/main/examples/spe10
[39] Qianchao Zhu and et al. 2021. Enabling and Scaling the HPCG Benchmark on the Newest Generation Sunway Supercomputer with 42 Million Heterogeneous Cores. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21). ACM, Article 57, 13 pages. https://doi.org/10.1145/3458817.3476158
[40] Yi Zong and et al. 2024. POSTER: StructMG: A Fast and Scalable Structured Multigrid. In Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, PPoPP 2024. ACM, 478–480. https://doi.org/10.1145/3627535.3638482