Performance Evaluation of Adaptive-Precision SpMV with Reduced-Precision Formats

Stef Graillat, Fabienne Jézéquel, Théo Mary, Roméo Molina, Daichi Mukunoki

Performance Evaluation of Adaptive-Precision SpMV with Reduced-Precision Formats

Stef Graillat[1], Fabienne Jézéquel[1,2], Théo Mary[1], Roméo Molina[1,3], and Daichi Mukunoki[4]

[1]Sorbonne Université, CNRS, LIP6, Paris, France
[2]Université Paris-Panthéon-Assas, Paris, France
[3]Université Paris-Saclay, Orsay, France
[4]RIKEN Center for Computational Science, Kobe, Japan

TABLE I
IEEE FORMATS AND RPFP'S REDUCED-PRECISION FORMATS

Data type	Data size (bits)	Format (bits) s:sign, e:exp, m:mantissa	Unit roundoff
binary64 (FP64)	64	s(1)+e(11)+m(52)	2^{-53}
FP64in56b (RP56)	32+16+8	s(1)+e(11)+m(44)	2^{-45}
FP64in48b (RP48)	32+16	s(1)+e(11)+m(36)	2^{-37}
FP64in40b (RP40)	32+8	s(1)+e(11)+m(28)	2^{-29}
binary32 (FP32)	32	s(1)+e(8)+m(23)	2^{-24}
FP32in24b (RP24)	16+8	s(1)+e(8)+m(15)	2^{-16}
binary16 (FP16)	16	s(1)+e(5)+m(10)	2^{-11}
FP32in16b (RP16)	16	s(1)+e(8)+m(7)	2^{-8}
FP16in8b (RP8)	8	s(1)+e(5)+m(2)	2^{-3}

***Abstract*—This study explores the potential for performance improvement in the adaptive-precision sparse matrix-vector product (SpMV) (Graillat et al. 2023) by using reduced-precision formats other than the IEEE standard ones. In addition to FP32 and FP64, we consider to utilize reduced-precision formats of 8/16/24/40/48/56-bit length. Our evaluation compares the performance of four- and nine-precision versions with that of the existing two-precision only version using FP32 and FP64 and demonstrates the effectiveness.**

***Index Terms*—sparse matrix-vector product (SpMV), mixed-precision, adaptive-precision, reduced-precision**

I. INTRODUCTION

The adaptive-precision sparse matrix-vector product (SpMV) [1] is a mixed-precision computation method for SpMV. It eliminates unnecessary bits on each element of the input matrix that do not contribute to the computed result at a target accuracy, and stores only the necessary bits with a possible minimum precision representation (or simply the element is dropped if all the bits are unnecessary). The reduced memory footprint is expected to result in faster execution time. The performance gain has so far been demonstrated only using the two-precision levels of IEEE FP32 and FP64 formats. In this study, through a reduced-precision memory accessor, we utilize some reduced-precision formats other than FP32 and FP64, such as 8/16/24/40/48/56-bit formats. Then, we demonstrate the performance with four- and nine-precision levels compared with the two-precision version on many-core CPUs.

Algorithm 1 Adaptive-precision SpMV $y = Ax$

```
for i = 1 : m do
    for k = 1 : q do
        y_i^(k) = 0
        for j ∈ B_ik do
            y_i^(k) = y_i^(k) + a_ij x_j in precision u_k
        end for
    end for
    y_i = Σ_{k=1}^{q} y_i^(k) in precision u_1
end for
```

II. METHODS

A. Adaptive-precision SpMV

The adaptive-precision SpMV decomposes the input matrix into several low-precision matrices of different precision formats and then computed the sum of the SpMVs for the decomposed low-precision matrices. Algorithm 1 shows the adaptive-precision SpMV computing $y = Ax$, where $A \in \mathbb{R}^{m\times n}$ and $x \in \mathbb{R}^n$, with q precision levels $u_1 < u_2 < ... < u_q$, where u_k $(k = 1, ..., q)$ denotes the unit roundoff of each precision used. Each row i of A is partitioned into q buckets B_{ik}, which is determined such that the backward error (see [1] for definition) is at most in $O(\epsilon)$, where $\epsilon \geq u_1$ is a target accuracy, as

$$B_{ik} = \{j \in J_i : |a_{ij}x_j| \in (\epsilon\theta_i/u_{k+1}, \epsilon\theta_i/u_k]\},$$

where J_i is the set of indices of nonzero elements in row i of A. For θ_i, $\theta_i = ||A||$ is used to satisfy the norm-wise error and $\theta_i = |a_i|^T e$, where $e = [1, ..., 1]^T$, is used to satisfy the component-wise error.

B. Reduced-precision formats

A reduced-precision memory accessor, RpFp [2], enables one to represent arbitrary precision mantissa in multiple bytes in C/C++ language by truncating the IEEE formats, as shown in Table I. For 24/40/48/56-bit formats, a reduced-precision value is represented as a structure with multiple words, and when allocating the array of the structure, an array is allocated for each word separately for efficient memory access (the

```
void ap_csrmv (int n, rpMultiCSR A, double* x, double* y) {
    #pragma omp parallel for
    for (int i = 0; i < n; i++) {
        double tmp = 0.;
        for (int k = A.ia8[i]; k < A.ia8[i+1]; k++) { // RP8
            float aij_r = RpArrayToFp(A.a8, k);
            tmp += (double)(aij_r * x[A.ja8[k]]);
        }
        for (int k = A.ia16[i]; k < A.ia16[i+1]; k++) { // RP16
            float aij_r = RpArrayToFp(A.a16, k);
            tmp += (double)(aij_r * x[A.ja16[k]]);
        }
    ...
        for (int k = A.ia56[i]; k < A.ia56[i+1]; k++) { // RP56
            double aij = RpArrayToFp(A.a56, k);
            tmp += aij * x[A.ja56[k]];
        }
        for (int k = A.ia64[i]; k < A.ia64[i+1]; k++) { // FP64
            double aij = A.a64[k];
            tmp += aij * x[A.ja64[k]];
        }
        y[i] = tmp;
    }
}
```

Fig. 1. Adaptive-precision SpMV with nine precision levels (a part). `RpArrayToFp` converts a reduced-precision format to the IEEE format.

TABLE II
TEST MATRICES (SORTED BY n_{nz}).

#	Matrix	n	n_{nz}
0	vas_stokes_4M	4,382,246	131,577,616
1	Cube_Coup_dt0	2,164,760	127,206,144
2	Flan_1565	1,564,794	117,406,044
3	Long_Coup_dt6	1,470,152	87,088,992
4	bone010	986,703	71,666,325
5	vas_stokes_2M	2,146,677	65,129,037
6	Hook_1498	1,498,023	60,917,445
7	RM07R	381,689	37,464,962

structure-of-array layout). To obtain an element from the array, the elements are taken from multiple separate arrays and concatenated into a value. We note that FP32in16b (RP16) is equivalent to the bfloat16 (BF16) format and that FP16in8b (RP8) is equivalent to the FP8 E5M2 format [3].

C. Implementation

We ported the Fortran code used in the paper [1] to the C language and applied the RpFp implementation for CPUs from [4] to it. The FP16 and RP8 formats rely on the half library[1]. Our implementation assumes that the input matrix fits into the 8-bit exponent range of FP32, but since FP16 and RP8 have only a 5-bit exponent, they are not used if the element is out of range. The SpMV adopts the Compressed Row Storage (CRS) format with 32-bit indexes. It is parallelized with OpenMP: a matrix row is computed with a thread. Figure 1 presents a part of the adaptive-precision SpMV code with nine precision levels, computing $y = Ax$.

III. EVALUATION

We performed the evaluation on the dual-socket system of AMD EPYC 7713 (Zen3 architecture). This processor has the following specifications (given for one CPU): 64 cores, AVX2 (256-bit SIMD), theoretical peak performance of 2048 GFlops/s in FP64 and 4096 GFlops/s in FP32 (at 2.0 GHz), 256 MB L3 cache, DDR4 memory of 204.8 GB/s. The

[1]https://sourceforge.net/projects/half/

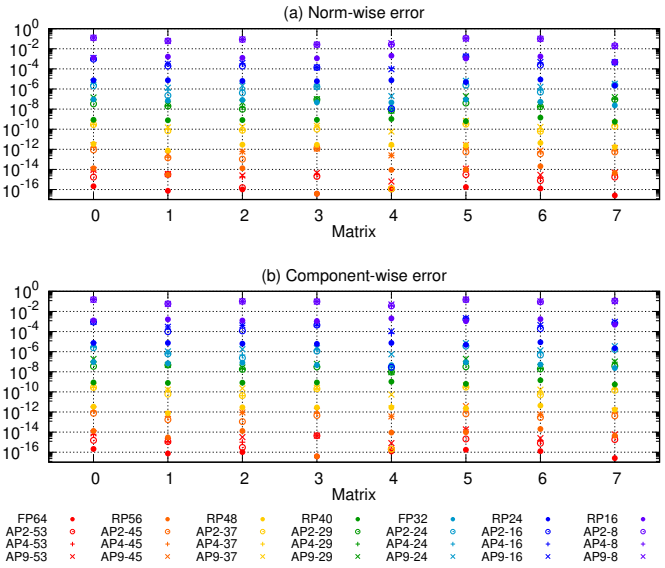

Fig. 2. Error obtained from the FP128 uniform precision SpMV

experimental code was compiled using GCC 9.4.0 with `-O3 -march=native -fopenmp -lgomp` (2 threads/core). It was executed with `numactl --interleave=all`. We collected eight potential matrices from the SuiteSparse Matrix Collection [5], as presented in Table II (each matrix has size of $n \times n$ with n_{nz} nonzero elements). These matrices were numbered in n_{nz} descending order. Symmetry is not considered in SpMV; the matrix is expanded to an asymmetric matrix before execution. The vector x is set to $e = [1, ..., 1]^T$. For the results, we reported the shortest execution time out of 15 executions (SpMV is executed five times in a single program, and the program is executed three times.).

We evaluated the following cases:

- **FPxx**: Uniform-precision SpMV with FPxx (xx=32 or 64).
- **RPxx**: Reduced-precision SpMV with RPxx (xx=8, 16, 24, 40, 48, or 56). The matrix and vectors are stored in RPxx while the arithmetic operations are performed in the IEEE type (FP32 or FP64) having the same exponent range as RPxx.
- **AP2**: Adaptive-precision SpMV with two precision levels using FP64 and FP32.
- **AP4**: Adaptive-precision SpMV with four precision levels using FP64, FP48, FP32, and RP16.
- **AP9**: Adaptive-precision SpMV with nine precision levels using FP64, FP56, FP48, FP40, FP32, FP24, FP16, RP16, and RP8.

Figure 2 presents the backward errors obtained from the uniform precision SpMV performed in FP128 arithmetic.

Figure 3 presents the relative execution time normalized to FP64. The constant p determines the target accuracy as $\epsilon = 2^{-p}$, which corresponds to the unit roundoff shown in Table I. AP4 and AP9 are better than AP2 in many cases. For example, the maximum speedup of AP4 over AP2 is 47% (matrix #1,

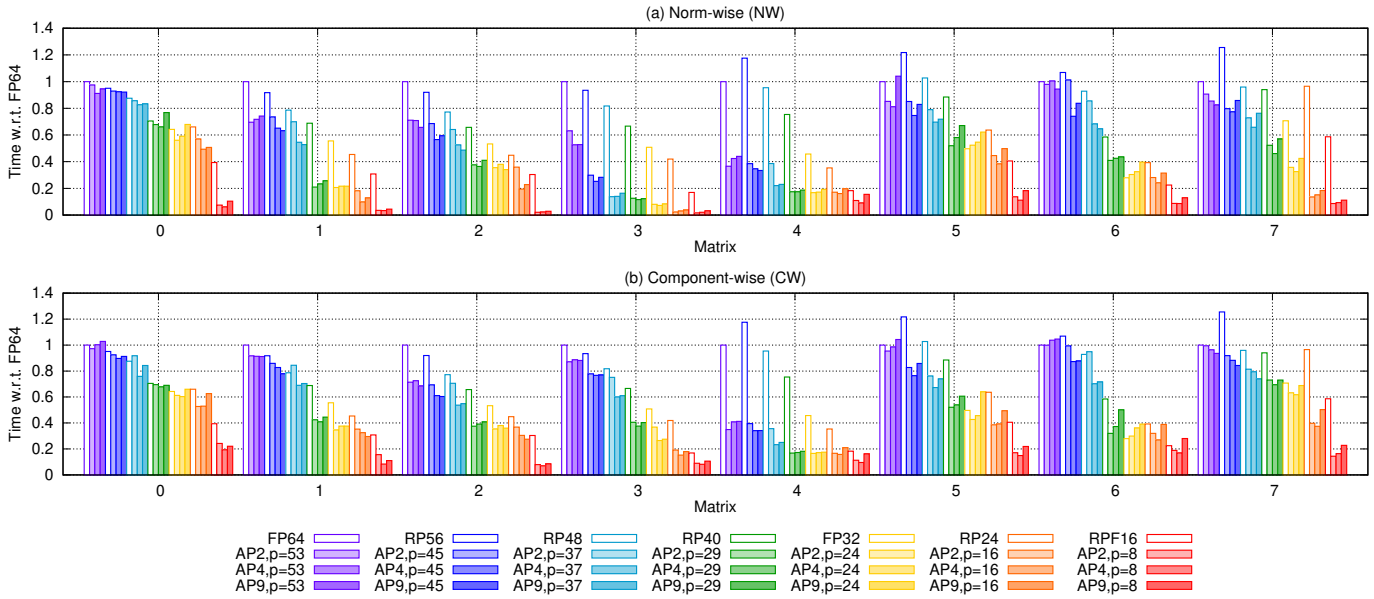

Fig. 3. Execution time (normalized to the time of the FP64 SpMV)

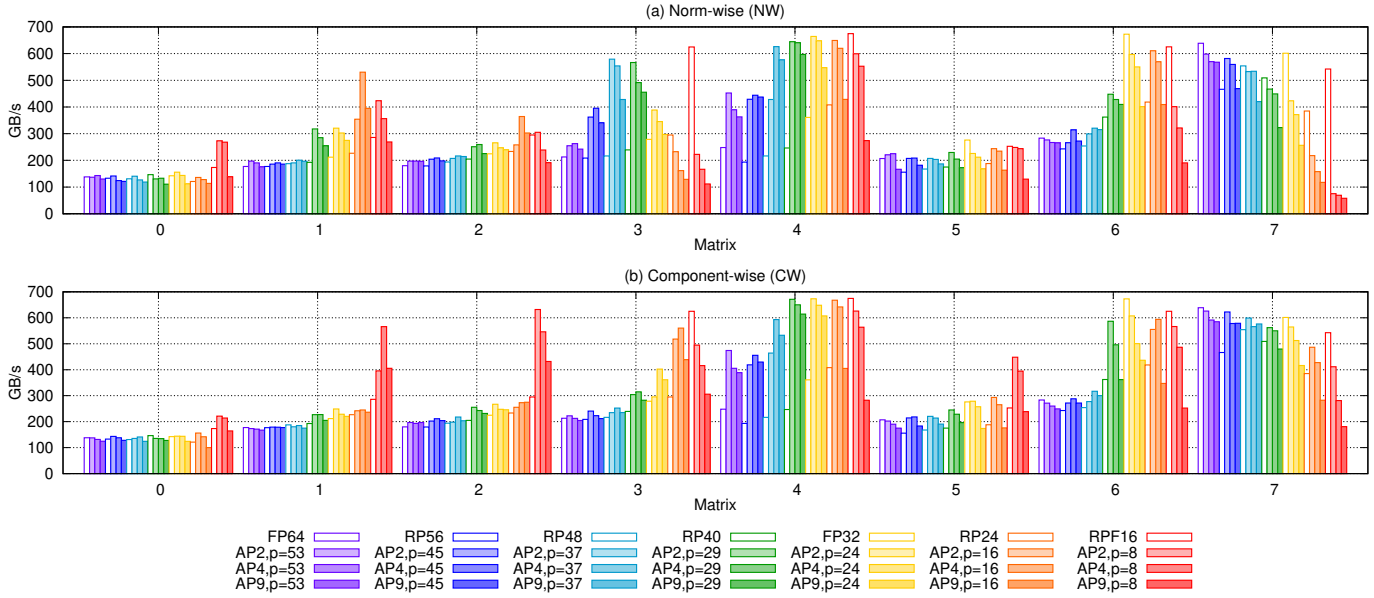

Fig. 4. Throughput in GB/s

CW, p=8). However, the difference between AP4 and AP9 is trivial, or rather AP4 is better in many cases. Still, we observed the maximum improvement of 9.7% for AP9 over AP4 (matrix #2, CW, p=16) among the cases where AP4 is faster than AP2. Figure 4 presents the throughput in GB/s. As the data size decreases, the throughput can increase due to cache (note that some exceed the memory bandwidth). Figure 5 shows the distribution of the formats used in the matrix, which explains that the performance gains come from the use of dropping and low-precision formats. The effectiveness is highly matrix-dependent, but in many cases where the target accuracy is not corresponding to the IEEE format, the introduction of reduced-precision formats has led to successful speedups.

IV. Conclusion

We have demonstrated the performance of adaptive-precision SpMV with up to nine-precision levels using 8/16/24/32/40/48/56/64-bit formats. Increasing the number of precision levels used does not necessarily improve performance, and the effectiveness is strongly dependent on the matrix and target accuracy, but the effect was demonstrated in several cases.

Below are future work and perspectives.

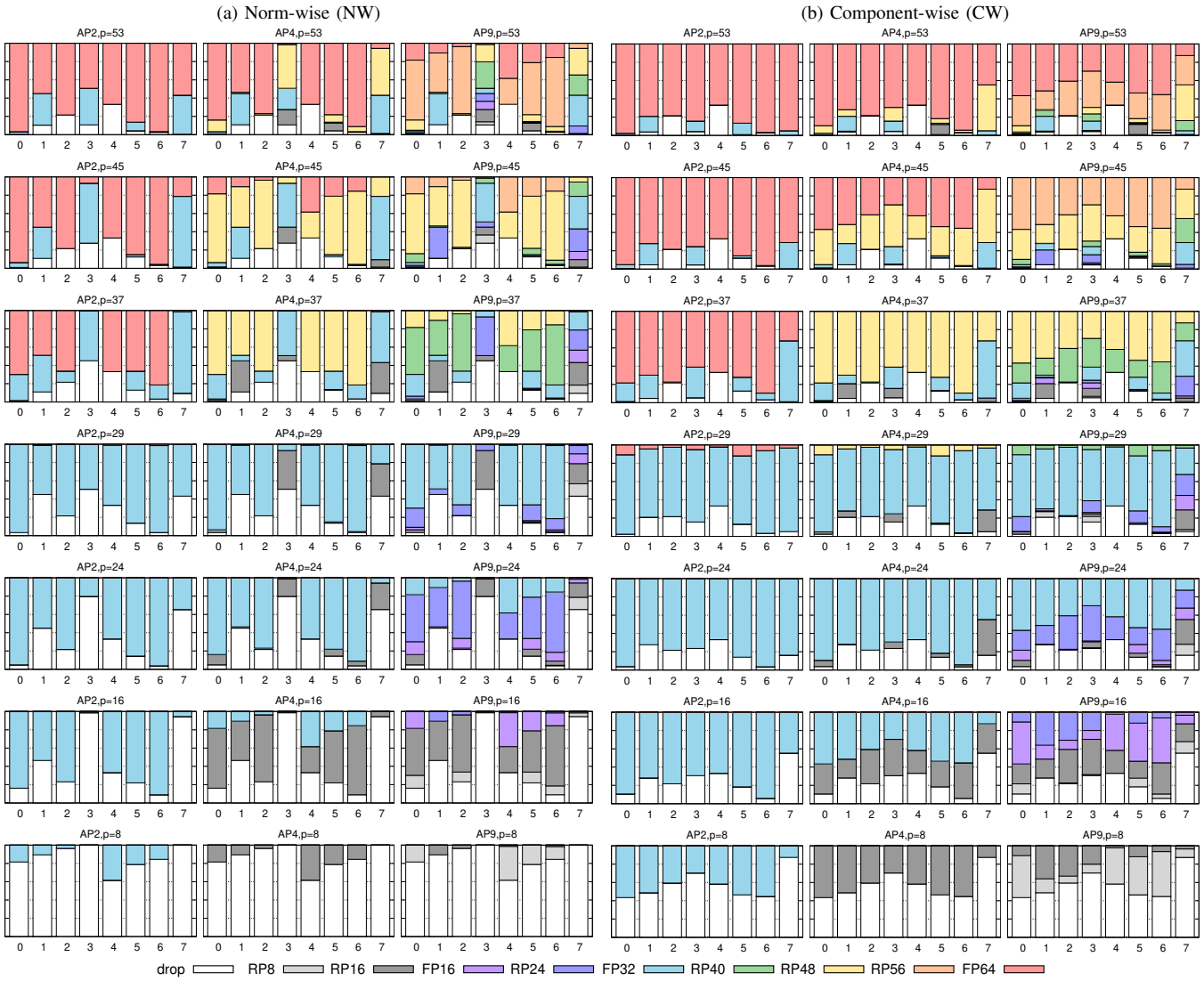

Fig. 5. Distribution of formats used in adaptive-precision SpMV. The horizontal axis indicates the matrix number and the vertical axis indicates the percentage (up to 100%).

- Since the use of low-precision formats does not necessarily lead to better performance, performance-oriented decisions on which formats to use can be explored (e.g., excluding RP56 that is composed of 3 elements).
- New sparse matrix formats suitable for adaptive-precision SpMV can be explored (e.g., index-compressed format is effective for low precision.).
- Compression of the exponent part or decomposition of the matrix according to the range of the exponent can be explored. Or, one can consider introducing a reduced format with a smaller exponential part, such as FP16 for BF16.

ACKNOWLEDGMENT

This research was supported by the Japan Society for the Promotion of Science (JSPS) KAKENHI Grant #20KK0259 and the InterFLOP (ANR-20-CE46-0009) project of the French National Agency for Research (ANR) and the interdisciplinary CNRS project CASSIDI.

REFERENCES

[1] S. Graillat, F. Jézéquel, T. Mary, and R. Molina, "Adaptive precision sparse matrix-vector product and its application to Krylov solvers," *SIAM Journal on Scientific Computing*, 2023, in press. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03561193

[2] D. Mukunoki and T. Imamura, "Reduced-Precision Floating-Point Formats on GPUs for High Performance and Energy Efficient Computation," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, 2016, pp. 144–145.

[3] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, N. Mellempudi, S. Oberman, M. Shoeybi, M. Siu, and H. Wu, "Fp8 formats for deep learning," 2022.

[4] D. Mukunoki, M. Kawai, and T. Imamura, "Sparse Matrix-Vector Multiplication with Reduced-Precision Memory Accessor," in *16th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC 2023)*, 2023, (accepted).

[5] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, dec 2011.