Mixed precision support in SparCity Framework

Deliverable No:	D2.2
Deliverable Title:	Mixed precision support in SparCity Framework
Deliverable Publish Date:	31 March 2023
Project Title:	SparCity: An Optimization and Co-design Framework for Sparse Computation
Call ID:	H2020-JTI-EuroHPC-2019-1
Project No:	956213
Project Duration:	36 months
Project Start Date:	1 April 2021
Contact:	sparcity-project-group@ku.edu.tr

List of partners:

Participant no.	Participant organisation name	Short name	Country
1 (Coordinator)	Koç University	KU	Turkey
2	Sabancı University	SU	Turkey
3	Simula Research Laboratory AS	Simula	Norway
4	Instituto de Engenharia de Sistemas e Computadores, Investigação e Desenvolvimento em Lisboa	INESC-ID	Portugal
5	Ludwig-Maximilians-Universität München	LMU	Germany
6	Graphcore AS	Graphcore	Norway

CONTENTS

1 INTRODUCTION

The SparCity project is funded by EuroHPC JU (the European High Performance Computing Joint Undertaking) under the 2019 call of Extreme Scale Computing and Data Driven Technologies for research and innovation actions. SparCity aims to create a supercomputing framework that will provide efficient algorithms and coherent tools specifically designed for maximizing the performance and energy efficiency of sparse computations on emerging High Performance Computing (HPC) systems, while also opening up new usage areas for sparse computations in data analytics and deep learning.

Sparse computations are commonly found at the heart of many important applications, but at the same time it is challenging to achieve high performance. SparCity delivers a coherent collection of innovative algorithms and tools for enabling high efficiency of sparse computations on emerging hardware platforms. More specifically, the objectives of the project are:

- to develop a comprehensive application and data characterization mechanism for sparse computation based on the state-of-the-art analytical and machine-learning-based performance and energy models,
- to develop advanced node-level static and dynamic code optimizations designed for massive and heterogeneous parallel architectures with complex memory hierarchy for sparse computation,
- to devise topology-aware partitioning algorithms and communication optimizations to boost the efficiency of system-level parallelism,
- to create digital SuperTwins of supercomputers to evaluate and simulate what-if hardware scenarios,
- to demonstrate the effectiveness and usability of the SparCity framework by enhancing the computing scale and energy efficiency of challenging real-life applications,
- to deliver a robust, well-supported and documented SparCity framework into the hands of computational scientists, data analysts, and deep learning end-users from industry and academia.

1.1 OBJECTIVES OF THIS DELIVERABLE

The objective of this deliverable is to document the ongoing research regarding the exploitation of lower precision units for sparse computation and development of techniques to boost the performance without significantly impacting accuracy of sparse solvers. The deliverable will also discuss the design of new storage formats to leverage mixed precision, along with their pros and cons.

1.2 WORK PERFORMED

The research work that has so far been carried out has focused on exploring mixed precision implementations for one of the most frequently used sparse kernels, namely SpMV (sparse matrix vector multiplication). We have developed a fine-grained mixed-precision SpMV, where the level of precision is decided for each element in the matrix to be used in a single operation. We extend an existing entry-wise precision based approach by deciding precisions per row, motivated by the granularity of parallelism on a GPU where groups of threads process rows in CSR-based

matrices. We propose mixed-precision CSR storage methods with row permutations and describe their greater efficiency and load-balancing compared to the existing method. We also consider a multi-precision case where single and double precision copies of the matrix are stored ahead of time, and further extend our mixed-precision SpMV approach to comply with it. As such, we leverage a mixed-precision SpMV to obtain a multi-precision Jacobi method which is faster than yet almost as accurate as double-precision Jacobi implementation, and further evaluate a multi-precision Cardiac modeling algorithm. We demonstrate the effectiveness of the proposed SpMV methods on an extensive dataset of real-valued large sparse matrices from the SuiteSparse Matrix Collection using an NVIDIA V100 GPU.

1.3 DEVIATIONS AND COUNTER MEASURES

There was no deviation from the work plan.

1.4 RESOURCES

- The source code repository for the mixed and multi-precision support for SparCity is public and available at

  https://github.com/sparcityeu/Codebase

- Most of the content in this deliverable also appears in the publication that came out from this work:

  Erhan Tezcan, Tugba Torun, Fahrican Koşar, Kamer Kaya, and Didem Unat (2022). Mixed and Multi-Precision SpMV for GPUs with Row-wise Precision Selection. IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'22), November 2-5, 2022, Bordeaux, France. (**Best Paper Award**)

2 MIXED AND MULTI-PRECISION SPMV

SpMV has been extensively researched with different storage formats and GPU kernel optimizations. One potential optimization is to utilize mixed-precision, combining double precision (FP64) and single precision (FP32). This is because lower precision arithmetic is faster and more power-efficient, with less memory and network traffic than higher precision arithmetic. Mixed-precision SpMV can be implemented at different levels of the algorithm, such as within iterative solvers or in compute-intensive parts of the application.[1]

A mixed-precision SpMV technique is to use different precisions at different levels of the algorithm within SpMV. This is commonly used in iterative solvers where a less-precise inner solver is kept at lower precision, and an outer, more-precise solver can tolerate the errors introduced, an approach known as defect-correction.[2] Lower precisions have also been utilized in compute-intensive but error-tolerant parts of iterative solvers, such as factorization and preconditioning.[3]

Another approach to mixed-precision SpMV is to use different precisions at a finer granularity within a single SpMV.[4] Ahmad et al.[5] use different precisions for each nonzero value of the sparse matrix stored in CSR format and split it into separate FP32 and FP64 CSR matrices. However, their method suffers from load-imbalance issues, resulting in slower execution times than FP64 SpMV. In this project, row-wise precision selection is proposed to balance the workload among threads and extend CSR and ELLPACK-R to multi-precision with slightly modified SpMV kernels.

The modified kernels are based on the open-source CUSP library[6] for NVIDIA, an implementation of CSR-Vector SpMV that has been used in numerous works comparing different kernels. Upon deciding the precision for each row, the rows are permuted such that those with the same precision type are clustered together to address load-balancing issues.

The proposed mixed-precision SpMV is applied to a multi-precision sparse Jacobi method and a multi-precision Cardiac modeling application[7] of solving diffusion equations with finite volumes. The main contributions of this work are the implementation of row-wise precision selection to balance the workload among threads and the demonstration of the capability of mixed-precision SpMV in different applications.

[1] Ahmad Abdelfattah et al. "A survey of numerical linear algebra methods utilizing mixed-precision arithmetic". *Int. J. High Perf. Comp. Appl.* 35.4 (2021), pp. 344–369; Jennifer A. Loe et al. "Experimental Evaluation of Multiprecision Strategies for GMRES on GPUs". *IEEE Int. Parallel Distrib. Process. Symp. Workshops*. 2021, pp. 469–478; Stephen F. McCormick, Joseph Benzaken, and Rasmus Tamstorf. "Algebraic error analysis for mixed-precision multigrid solvers". *ArXiv* abs/2007.06614 (2020).

[2] Erin Carson and Nicholas J. Higham. "Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions". *SIAM J. Sci. Comput.* 40.2 (2018), A817–A847; M.A. Clark et al. "Solving lattice QCD systems of equations using mixed precision solvers on GPUs". *Comput. Phys. Commun.* 181.9 (2010), pp. 1517–1528. ISSN: 0010-4655.

[3] Ahmad Abdelfattah, Stan Tomov, and Jack Dongarra. "Investigating the Benefit of FP16-Enabled Mixed-Precision Solvers for Symmetric Positive Definite Matrices Using GPUs". *Comput. Sci.* Springer Int. Publ., 2020, pp. 237–250; Marc Baboulin et al. "Accelerating scientific computations with mixed precision algorithms". *Comput. Phys. Commun.* 180.12 (2009), pp. 2526–2533. ISSN: 0010-4655; Alfredo Buttari et al. "Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance While Achieving 64-Bit Accuracy". *ACM Trans. Math. Softw.* 34.4 (2008). ISSN: 0098-3500.

[4] Khalid Ahmad, Hari Sundar, and Mary Hall. "Data-Driven Mixed Precision Sparse Matrix Vector Multiplication for GPUs". *ACM Trans. Archit. Code Optim.* 16.4 (2019). ISSN: 1544-3566; Patrick Amestoy et al. "Mixed Precision Low Rank Approximations and their Application to Block Low Rank LU Factorization". hal-03251738v2. 2021; Rise Ooi et al. "Effect of Mixed Precision Computing on H-Matrix Vector Multiplication in BEM Analysis". *Proc. Int. Conf. High Perf. Comput. Asia-Pacific Region*. Fukuoka, Japan: ACM, 2020, pp. 92–101.

[5] Ahmad, Sundar, and Hall, "Data-Driven Mixed Precision Sparse Matrix Vector Multiplication for GPUs".

[6] Steven Dalton et al. *Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations*. Version 0.5.0. 2014. URL: http://cusplibrary.github.io/.

[7] Johannes Langguth et al. "Scalable Heterogeneous CPU-GPU Computations for Unstructured Tetrahedral Meshes". *IEEE Micro* 35.4 (2015), pp. 6–15.

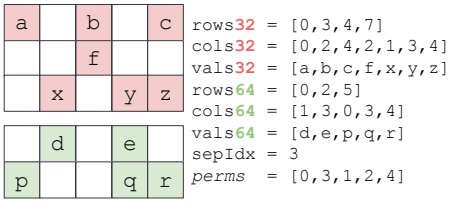


Figure 1 *Row-wise Split example*

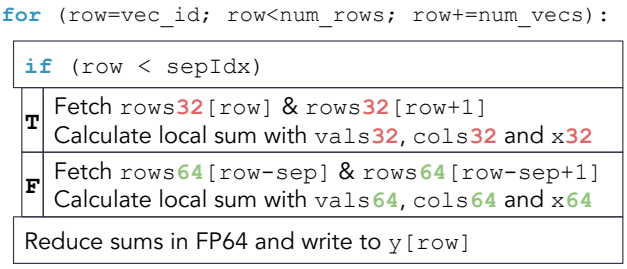


Figure 2 *Row-wise Split kernel*

2.1 ROW-WISE SPLIT: A MIXED PRECISION METHOD

We propose a row-level precision selection method for SpMV, namely *row-wise split*, which determines the precision for each row separately. A row is chosen to be in FP32 if at least p percentage of its values are in some range $(-r, r)$, and all of its values are within the range of FP32.

We also propose a *matrix-specific range* calculated for each matrix prior to computations, instead of a fixed range. This is because although a fixed and small range would incur less absolute error, it can cause all values to be stored in FP32 for matrices with small or scaled values. For a given matrix with a set of nonzero values $\mathcal{V}$, we set the range as $r = f \times (\sum_{v \in \mathcal{V}} |v|)/|\mathcal{V}|$ where f is a predefined shrinking factor chosen to adjust the range such that smaller f may lead to less error but fewer values to be stored in FP32.

After precision is decided per row, two matrices are created from the rows, one for each type of precision. This has a row permutation effect (shown with `perms` in Fig.1) where the rows with the same precision are clustered together. As shown in the kernel overview in Fig. 2, a thread group checks the precision by comparing its row to the separation index (`sepIdx`), and proceeds with the dot product at the respective precision.

The row-wise split method guarantees no empty rows, by permuting the existing empty rows to be clustered at the bottom and ignored during SpMV. This saves redundant global accesses to the row pointer in the presence of empty rows. It also effectively solves the load-balance problems. As depicted in Fig. 1, row permutation ensures consecutive thread groups to have similar loads. Moreover, a thread group's load does not change after the split as elements of a row stay the same.

2.2 ROW-WISE COMPOSITE: A MULTI-PRECISION COMPLIANT METHOD

Multi-precision and mixed-precision techniques are occasionally used interchangeably.[8] In this work, mixed-precision refers to the case that an operation uses multiple precisions; whereas multi-precision means that FP64, FP32 and mixed-precision implementations may be used throughout an application at different levels. For instance, a multi-precision GMRES method using FP32 and

[8] Abdelfattah et al., "A survey of numerical linear algebra methods utilizing mixed-precision arithmetic".

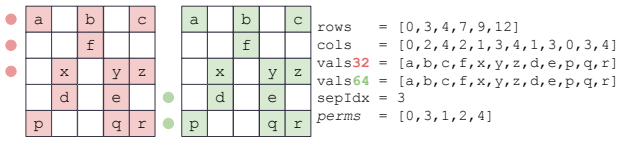


Figure 3 *Row-wise Composite Split example*

```
for (row=vec_id; row<num_rows; row+=num_vecs):
```

Fetch `rows[row]`, `rows[row+1]`	
`if (row < sepIdx)`	
T	Calculate local sum with `vals32`, `cols` and `x32`
F	Calculate local sum with `vals64`, `cols` and `x64`
Reduce sums in FP64 and write to `y[row]`	

Figure 4 *Row-wise Composite kernel*

FP64 is recently described,[9] and this method requires the matrix to be stored in both precisions at the same time. Note that the extra FP32 matrix just requires a value pointer, as the row and column pointers are equivalent for both matrices.

Suppose one is to extend the multi-precision method by incorporating mixed-precision SpMV in between FP32 and FP64 steps. The aforementioned methods need two different matrices with their own row, column and value pointers, differing from those of FP64 and FP32 matrices. Consequently, these mixed-precision splits require their own memory allocations and transfers for a GPU-based execution in addition to the existing FP32 and FP64 matrices. This can either mean allocating all types once at the start or transferring the data to GPU during execution, which would greatly hinder the performance due to large amounts of data movement.

To address this issue, we propose *row-wise composite* to be used in multi-precision cases. Instead of creating two matrices as done in row-wise split, here we just create an FP32 copy of the values array of the permuted FP64 matrix. Fig. 3 and Fig. 4 show how this enables accessing the value pointer in both precisions with the same row and column pointers. Similar to the row-wise split kernel, the separation index (`sepIdx`) is used to find the precision of a row.

In Table 1, we present the total storage cost and bandwidth requirement of each described mixed-precision split in bytes. Compared to the prior art, the row-wise split saves 4M bytes as the total number of rows stays constant. Row-wise composite has the cost of storing FP32 and FP64 CSR matrices together (with common row and column pointers) will cost 4M+16V+4 bytes. The bandwidth requirements of row-wise split and row-wise composite are equivalent for mixed-precision SpMV, and they save 4M compared to the prior art.

We extend our mixed-precision approach for the ELLPACK-R[10] format. The value and column index arrays have a size equal to number of nonzeros in CSR, but here the size is $M \times R$, where M is the row count and R is the maximum number of nonzeros in a row. This enables to index a row i just with $R \times i$. The row pointers in CSR become row lengths for ELLPACK-R, where index i is the number of nonzeros in row i. Consequently, the SpMV kernels are almost identical.

Provided that the number of nonzeros per row does not vary dramatically throughout the matrix, ELLPACK-R provides better memory coalescing and is more efficient than CSR format.

[9] Loe et al., "Experimental Evaluation of Multiprecision Strategies for GMRES on GPUs".

[10] F. Vázquez, J. J. Fernández, and E. M. Garzón. "A New Approach for Sparse Matrix Vector Product on NVIDIA GPUs". *Concurr. Comput.* 23.8 (2011), pp. 815–826. ISSN: 1532-0626.

Table 1 *Memory Capacity and Bandwidth Requirements.*

Storage Format	Storage Cost (Bytes)*	Bandwidth Requirement (Bytes)†
CSR FP64	$4M+12V+4$	–
CSR FP32	$4M+8V+4$	–
Prior Art	$8M+8V+4V_{64}+8$	$8M+8V+4V_{64}+8$
Row-wise Split	$4M+8V+4V_{64}+12$	$4M+8V+4V_{64}+12$
Row-wise Composite	$4M+16V+8$	$4M+8V+4V_{64}+12$

* M: #rows, V: #nonzeros, V_{64}: #nonzeros stored in FP64. † bandwidth required by the kernel

However, if a row has a lot more nonzeros compared to the others, this becomes memory-consuming. The splitting methodology is analogous to what is done for CSR format, the rows are treated the same way and two new ELLPACK-R matrices are created from the split.

2.3 CASE STUDY: JACOBI METHOD

Jacobi is a well-known iterative method that can be used to solve linear systems,[11] and is most widely used as a preconditioner within other iterative solvers (e.g., Krylov subspace methods) for faster convergence.[12] Using Jacobi as a preconditioner is appealing since its structure it allows utilizing high levels of parallelism in contrast to its counterparts such as ILU and SSOR.[13] The method aims to obtain an approximate solution for a linear system of equations $Ax = b$ where $A \in \mathbb{R}^{N\times N}$ is a sparse nonsingular matrix. Considering A is decomposed as $A = D+R$ where D is the diagonal component and R consists of the remaining off-diagonal entries, it iteratively solves $x^{(k+1)} = D^{-1}(b-Rx^{(k)})$ where $x^{(k)}$ is the solution at the k^{th} iteration. The SpMV operation in Jacobi ($Rx^{(k)}$) is required at each iteration, dominates the runtime, and constitutes the main bottleneck. We propose to utilize the mixed-precision SpMV to obtain a mixed-precision Jacobi method which is faster than the conventional FP64 Jacobi while attaining a decent accuracy.

The pseudocode of the proposed mixed-precision Jacobi method is given in Algorithm 1. Here x_{64} and x_{32} represent the solution vector x stored in FP64 and FP32, respectively. The R matrix is split as $R_{32}+R_{64}$, where R_{32} and R_{64} contain the nonzeros which are selected to be stored in FP32 and FP64, respectively. In Line 3, the mixed-precision SpMV is performed. In practice, instead of storing D as an $N\times N$ matrix, the diagonal array of D is stored in a vector d (in FP64).

We implement a single CUDA kernel for the Jacobi iteration and solution updates (lines 4, 5) where thread i computes $(b[i]-\bar{x}[i])/d[i]$ and stores it to register. It then writes this to $x_{64}[i]$ and $x_{32}[i]$, while casting the value to `float` for the latter. Although there is a slight overhead of updating the FP32 solution in addition to FP64 solution, the speedup gained from using mixed-precision SpMV is expected to amortize it. All computations take place on the GPU and there is no extra cost of transferring the solution vectors back and forth to the host. As described in the row-wise split and row-wise composite methods, the rows of the coefficient matrix are permuted prior to the computations. In the Jacobi method, the columns are further permuted in the same order with rows, i.e. a symmetric permutation is applied to make the diagonal values remain on

[11] Phanisri P Pratapa, Phanish Suryanarayana, and John E Pask. "Anderson acceleration of the Jacobi iterative method: An efficient alternative to Krylov methods for large, sparse linear systems". *J. Comput. Physics* 306 (2016), pp. 43–54; Ananda D Gunawardena, SK Jain, and Larry Snyder. "Modified iterative methods for consistent linear systems". *Linear Algebra Appl.* 154 (1991), pp. 123–143.

[12] Eli Turkel. "Preconditioning techniques in computational fluid dynamics". *Annu. Rev. Fluid Mech.* 31.1 (1999), pp. 385–416; SH Chan, KK Phoon, and FH Lee. "A modified Jacobi preconditioner for solving ill-conditioned Biot's consolidation equations using symmetric quasi-minimal residual method". *Int. J. Numer. Anal. Methods. Geomech.* 25.10 (2001), pp. 1001–1025; F-H Lee et al. "Performance of Jacobi preconditioning in Krylov subspace solution of finite element equations". *Int. J. Numer. Anal. Methods. Geomech.* 26.4 (2002), pp. 341–372.

[13] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

Require: Diagonal (D) and remaining (R) components with splitting $R = R_{32} + R_{64}$

1: Choose an initial guess $x^{(0)}$
2: **for** $k = 1, 2, ...,$ `ITERS` **do**
3: $\quad \bar{x} \leftarrow R_{64} \times x_{64}^{(k-1)} + R_{32} \times x_{32}^{(k-1)}$
4: $\quad \bar{x} \leftarrow D^{-1}(b - \bar{x})$
5: $\quad x_{64}^{(k)} \leftarrow \bar{x}, ;$
$\quad\quad x_{32}^{(k)} \leftarrow$ `float(`$\bar{x}$`)`
6: **end for**

Algorithm 1: Mixed-Precision Jacobi Method

Require: Diagonal (D) and remaining (R) components with splitting $R = R_{32} + R_{64}$

1: Initial guess $x^{(0)}$ read from disk.
2: **for** $k = 1, 2, ...,$ `ITERS` **do**
3: $\quad \bar{x} \leftarrow R_{64} \times x_{64}^{(k-1)} + R_{32} \times x_{32}^{(k-1)}$
4: $\quad \bar{x} \leftarrow \bar{x} + \bar{x} \odot D$
5: $\quad x_{64}^{(k)} \leftarrow \bar{x}, ;$
$\quad\quad x_{32}^{(k)} \leftarrow$ `float(`$\bar{x}$`)`
6: **end for**

Algorithm 2: Mixed-Precision Cardiac Modeling

the diagonal after reordering.

Different multi-precision implementations are possible for the Jacobi method. In a 1-step method, only the mixed-precision Jacobi is used. In a *2-step* Jacobi method, it starts with mixed-precision and moves to FP64 after a number of iterations. Similarly, in a *3-step* method, a number of iterations are done in FP32, then the method is upgraded to use mixed-precision, and the final set of iterations are done in FP64.

2.4 CASE STUDY: CARDIAC MODELING

In cardiac modeling, diffusion equations with finite volumes are used, where the computations for a single time-step can be shown as a matrix multiplication operation $x^{(i)} = A \times x^{(i-1)}$.[14] The mixed precision SpMV is utilized in a similar fashion to the Jacobi, where a solution time-step becomes $\bar{x} = A_{64} \times x_{64}^{(i)} + A_{32} \times x_{32}^{(i)}$. We then update both $x_{64}^{(i)}$ and $x_{32}^{(i)}$ solution vectors by assigning $\bar{x}$. Generally, the solution update could be made just by pointer swapping; however, when we have two solution vectors to be updated, swapping alone does not suffice. Instead, an additional copy kernel that is issued after the SpMV must update both solutions. We have also observed that the diagonal values in this dataset are much larger than off-diagonals. Our absolute mean based range-selection method would therefore end up with a large range that covers most of, if not all, the off-diagonal values; the only diagonal value which is outside the range in that row would be tolerated by the 1% margin. Consequently the whole matrix would likely be stored in FP32.

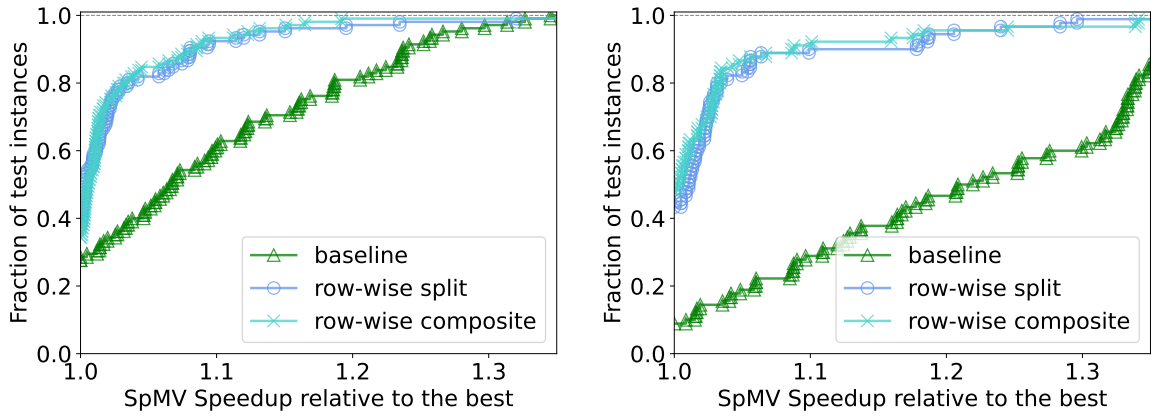
Considering both the large diagonals, and the need of an extra copy kernel to update solution vectors, we extract the diagonal values and store them in an array; whereas the off-diagonal values are kept in the matrix. After SpMV, the copy kernel adds the product of solution vector and diagonal values at their respective rows, and assigns the result to both FP64 and FP32 solution vectors as described in Algorithm 2. By utilizing the row-wise composite method, we also propose multi-precision Cardiac modeling methods *1-step*, *2-step* and *3-step* similar to the way that we construct in multi-precision Jacobi.

3 EVALUATION

3.1 SPMV RESULTS

For SpMV, we consider $y \leftarrow y + Ax$ where the initial y is all zeros, and the x entries are (uniformly) random in the range $(-5, 5)$. The experiments are conducted on real-valued matrices from the

[14] Langguth et al., "Scalable Heterogeneous CPU-GPU Computations for Unstructured Tetrahedral Meshes".



(a) *CSR* **(b)** *ELLPACK-R*

Figure 5 *SpMV Speedup performance profiles*

SuiteSparse Collection,[15] where the matrices have between 100K and 40M nonzeros. The range value of the baseline method is $r=1$, as in Ref..[16] For reliable measurements, we filter out matrices whose FP64 SpMV runtime is less than 0.1 seconds. We also ignore the matrices where the row-wise split stores less than 10% of nonzeros in FP32, as the speedup will not be considerable. As such, we have used 105 matrices (out of 2,794).

Fig. 5a shows the performance profiles comparing different methods in terms of the relative speedup of SpMV CSR with respect to FP64 for 105 matrices. A performance profile[17] represents the comparison of different methods for each data instance relative to the best-performing one. A point (α, β) on the line associated to method X means that the performance of X is within α factor of the best result for a fraction β of the instances. For example, the point (1.30, 0.75) on the curve of method X means that X yields 30% better result than the best result achieved for 75% of the dataset. Therefore, the method closest to the top left corner is interpreted as the best-performing one.

As shown in Fig. 5a, our proposed row-wise split and row-wise composite methods are performing the best. Due to their close performance, further SpMV experiments are conducted with the row-wise split method. Fig. 6 shows the plot for relative residuals of SpMV with respect to FP64 for the same 105 matrices sorted in the order of the baseline method's residuals. Here, we refer the relative residual as $\|y'-y_{64}\|/\|y_{64}\|$ where y_{64} is the result of FP64 SpMV, and y' is the result of mixed-precision SpMV. FP32 SpMV with FP64 reduction serves as the upper bound of relative residual, which is incurred by storing all nonzeros in FP32.

With its small fixed range $(-1, 1)$, the baseline often has small relative residuals. Nevertheless, there are numerous instances where the row-wise split yields near or lesser residual compared to the baseline. Considering the greater speedup of the proposed method and the possibility of correcting errors with a more precise iteration that follows the mixed-precision steps, our residuals to be acceptable.

For the chosen 105 matrices, ELLPACK-R goes out-of-memory for 15 of them, as ELLPACK-R stores a dense matrix of size $M \times R$ for a matrix with M rows and maximum of R elements in a

[15] Timothy A. Davis and Yifan Hu. "The University of Florida Sparse Matrix Collection". *ACM Trans. Math. Softw.* 38.1 (2011). ISSN: 0098-3500.

[16] Ahmad, Sundar, and Hall, "Data-Driven Mixed Precision Sparse Matrix Vector Multiplication for GPUs".

[17] Elizabeth D Dolan and Jorge J Moré. "Benchmarking optimization software with performance profiles". *Math. Prog.* 91.2 (2002), pp. 201–213.

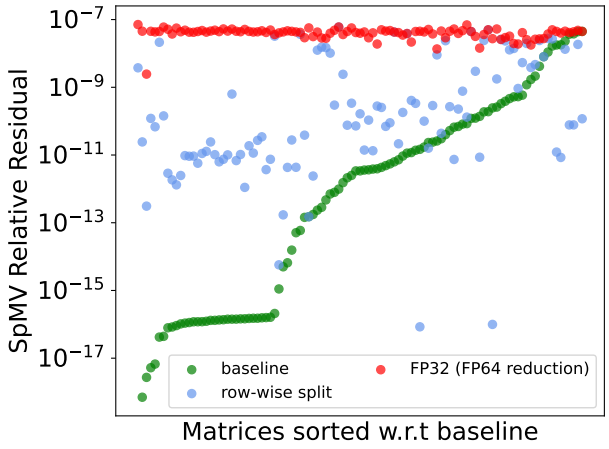


Figure 6 *SpMV CSR Relative Residuals*

Table 2 *Average speedup and relative residuals with respect to the FP64 SpMV.*

density (λ) in 10,000 *	matrix count	avg. speedup w.r.t. FP64			avg. relative residual w.r.t. FP64		
		FP32†	baseline	row-split§	FP32†	baseline	row-split§
λ <0.5	26	1.14	0.97	1.03	3.90e-08	8.31e-12	1.03e-09
0.5 <λ <1	16	1.13	1.02	1.06	3.68e-08	7.22e-12	6.53e-10
1 <λ <2.5	17	1.11	0.95	1.05	4.14e-08	8.79e-12	1.52e-11
2.5 <λ <5	24	1.13	0.98	1.07	3.30e-08	8.46e-12	6.81e-11
λ >5	22	1.26	1.10	1.11	4.49e-08	7.37e-12	4.19e-11
ALL	105	1.16	1.00	1.06	3.87e-08	8.04e-12	1.33e-10

* density $\lambda = \mathrm{NNZ}/(M \times N) \times 10{,}000$. † FP32 SpMV with FP64 sum reduction. § row-wise split.

row. If a row has many elements, but the rest are a lot sparser, this format becomes inefficient. The speedup results for the remaining 90 matrices are given in Fig. 5b. We omit ELLPACK-R relative residuals as they are invariant to the storage format and they are similar to what is observed with CSR. Moreover, the average and maximum speedup for ELLPACK-R is higher than that of CSR, with 1.07x and 1.74x respectively. Note that FP32 with double sum-reduction serves as an upper bound for speedup with average 1.16× and 1.13× for CSR and ELLPACK-R respectively.

In Table 2, we present the (geometric) average speedup and relative residuals grouped into five different matrix density categories of 105 matrices. For the row-wise split method, a density-speedup correlation is observed. Though its reason is not immediately obvious, the sparser matrices may have lower speedups as the irregular access to the dense vector becomes more pronounced. Note that the access pattern is not within the scope of mixed-precision SpMV nor our row permutations. FP32 with double sum reduction, serving as an upper bound of speedup and relative residual for each group, achieves up to 26% improvement with 16% on average. The table also shows that the baseline has no or insignificant speedup. We have also experimented on 2,794 real-valued matrices from SuiteSparse on which the experiments of Ref.[18] are conducted, and from these, in 999 matrices, row-wise split has speedup $\geqslant 1$; whereas this number is 354 for the baseline method, contrary to the findings in Ref..[19]

[18] Ahmad, Sundar, and Hall, "Data-Driven Mixed Precision Sparse Matrix Vector Multiplication for GPUs".
[19] Ibid.

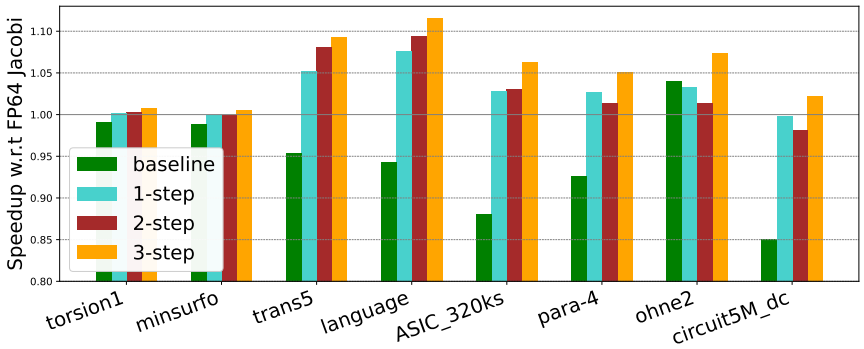


Figure 7 *Speedups for Jacobi method*

3.2 JACOBI METHOD RESULTS

We consider large real-valued sparse square matrices with no zero values in the diagonal. We applied HSL MC64[20] permutation and scaling for diagonal-dominance, which is a sufficient condition for Jacobi convergence.[21] Permuting aims to maximize the product of the diagonal entries, and scaling aims to make the absolute value of diagonal entries 1 and the others not larger than 1. We use $r = 0.01$ as a range for the baseline method (Jacobi using SpMV with entry-wise split) since otherwise our method would choose everything in FP32 after scaling.

We set the desired solution vector as $x^* \leftarrow [1N, 2N, \ldots, 1]^T$ to find $y \leftarrow Ax^*$ with FP64 SpMV. We set the initial guess as $x \leftarrow [0,0,\ldots,0]^T$ and solve $Ax = y$ using Jacobi. Since Jacobi is often used as a preconditioner, and to provide an explicit accuracy comparison, we run Jacobi for a fixed number (2,000) of iterations. The iterations are divided equally among the steps in multi-precision runs. The relative residual is calculated as $\|y - Ax'\|/\|y\|$ where x' is the computed solution. The comparison experiments are conducted on 8 matrices whose relative residual for FP64 Jacobi after 2000 iterations is less than 10^{-1}.

Fig. 7 shows the speedups of baseline and our methods compared to the FP64 Jacobi. In all cases, 3-step multi-precision Jacobi is the fastest (avg. 5% and up to 11% speedup) due to the performance gains from the lower precision steps. 2-step method also has speedup, but for some cases less than 1-step; possibly due to the additional cost of updating the FP32 copy of the solution vector, and low number of FP32 values used in split. For all cases, baseline is the slowest except for *ohne*, where the baseline keeps considerably high number of FP32 values.

Table 3 shows the relative residuals for FP64 and FP32 Jacobi, the baseline, and our methods. The residual is affected by the precision selection, but not from the split that comes afterward, since the split itself does not change the accuracy of the computations. Although the 1-step method has relative residuals higher than that of FP64, the ones of 2-step and 3-step are close to FP64 since they can tolerate the accumulated errors in the prior less-precise steps, as expected. This is in line with our expectations that lower precision steps can be corrected by the following higher precision steps.

[20] *HSL. A collection of Fortran codes for large scale scientific computation. http://www.hsl.rl.ac.uk/.*

[21] Roberto Bagnara. "A unified proof for the convergence of Jacobi and Gauss–Seidel methods". *SIAM Review* 37.1 (1995), pp. 93–97.

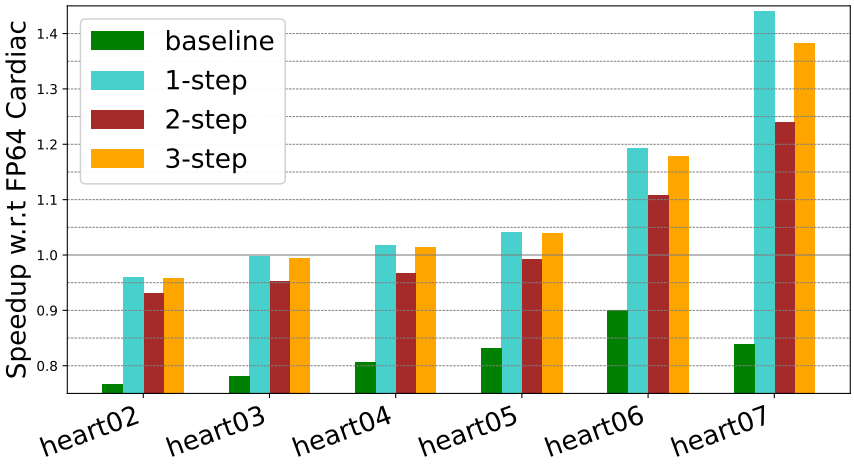


Figure 8 *Speedups for Cardiac modeling*

Table 3 *Relative residuals of Jacobi method after 2,000 iterations.*

matrix	FP64	FP32*	baseline	1-step	2-step	3-step
torsion1	1.49e-16	1.24e-07	1.49e-16	1.38e-08	1.58e-16	1.58e-16
minsurfo	1.19e-16	1.97e-07	1.40e-10	5.33e-09	2.27e-16	2.44e-16
trans5	1.07e-05	1.06e-05	1.07e-05	1.07e-05	1.07e-05	1.07e-05
language	7.14e-18	1.04e-08	1.59e-11	7.28e-09	7.16e-18	7.15e-18
ASIC_320ks	4.89e-07	4.89e-07	4.89e-07	4.89e-07	4.89e-07	4.89e-07
para-4	7.68e-02	7.68e-02	7.68e-02	7.68e-02	7.68e-02	7.68e-02
ohne2	2.12e-02	2.12e-02	2.12e-02	2.12e-02	2.12e-02	2.12e-02
circuit5M_dc	6.01e-17	3.25e-08	5.36e-11	6.34e-09	6.39e-17	6.68e-17

* FP32 Jacobi refers to Jacobi using FP32 SpMV with FP64 reduction.

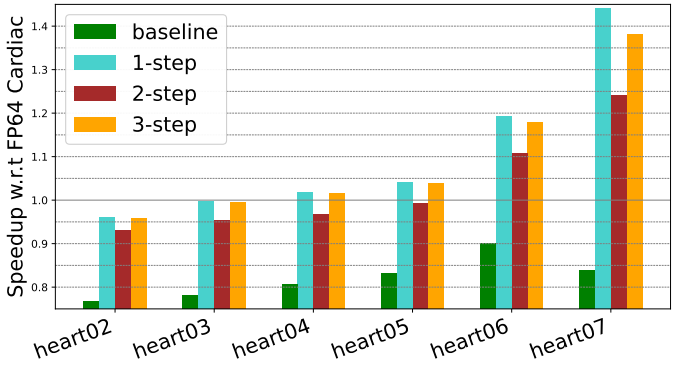


Figure 9 *Cardiac Speedups with respect to FP64 Cardiac method*

3.3 CARDIAC MODELING RESULTS

We use a set of 6 generated meshes (matrices) from Ref.[22] each having larger than 100K nonzeros and an initial dense solution vector. The ground-truth FP64 Cardiac modeling does not extract the diagonal, and uses pointer swapping to update the solution. Each method runs for 8,000 iterations. All off-diagonals for this dataset are less than 1, so again the baseline range $r = 1$ is infeasible, instead we use $r = 1e{-}5$, which is close to the absolute mean value for all 6 matrices.

[22] Hector Martinez-Navarro et al. *Repository for modelling acute myocardial ischemia: simulation scripts and torso-heart mesh, University of Oxford*. 2019.

Figure 9 shows the speedups of baseline and our methods with respect to the FP64 Cardiac modeling for 6 matrices sorted in increasing order of number of nonzeros. Despite the extra copy kernel overhead, 1-step achieves an average of 1.11× and up to 1.40× speedup, while also being faster than 2-step and 3-step. This indicates that FP64 steps are slow enough to affect the entire execution time. The speedup increases with the number of nonzeros, which may be due to better cache utilization of lower precision. The relative residuals are computed similar to the SpMV case, and are found to be not much different among methods (unlike Jacobi), thus are omitted for brevity.

4 CONCLUSIONS

Deliverable 2.2 focused on presenting the main contributions SparCity has made in terms of mixed precision support in sparse computations. Our main contributions are:

- We propose an easy mixed-precision method for SpMV and its CSR and ELLPACK-R based storage formats and GPU kernels.
- For a multi-precision setting where FP32 and FP64 matrices are stored in advance, we further extend the row-wise mixed-precision SpMV and implement a multi-precision Jacobi pre-conditioner and Cardiac modeling as use-cases.
- Over a dataset of 105 real-valued large matrices from SuiteSparse, for the CSR format, we demonstrate an average 1.06× and up to 1.49× speedup over FP64 SpMV, while the prior-art[23] achieves no speedup. FP32 SpMV with FP64 sum-reduction achieves an average speedup of 1.16×, which serves as an upper bound for the achievable speedup.
- For ELLPACK-R, we achieve an average 1.07× and up to 1.74× speedup, which is again very close to the upper bound of FP32 SpMV (with FP64 sum-reduction) with an average speedup of 1.13×.
- We demonstrate that multi-precision Jacobi is faster than, yet as accurate as, the FP64 Jacobi method. We achieve an average 1.05× and 1.11× and up to 1.11× and 1.40× speedup for Jacobi and Cardiac modeling, respectively.

Our methodology is based on CSR format; nevertheless, we demonstrate its effectiveness with ELLPACK-R, and we further believe that the same methodology will work on various other formats (e.g., CSC, DIA). Since FP16 (half-precision) is increasingly becoming prevalent among HPC and ML, it is a clear direction of future work to extend our methodology to accommodate lower- precisions. A further AMD-GPU extension of this work, although an orthogonal direction of optimization, is certainly possible. Scenarios where each GPU works on a different precision, or where the split matrices are further split to fit into multiple GPUs are of consideration; however, effort must go into load-balancing the work among them.

[23] Ahmad, Sundar, and Hall, "Data-Driven Mixed Precision Sparse Matrix Vector Multiplication for GPUs".

REFERENCES

Abdelfattah, Ahmad et al. "A survey of numerical linear algebra methods utilizing mixed-precision arithmetic". *Int. J. High Perf. Comp. Appl.* 35.4 (2021), pp. 344–369.

Abdelfattah, Ahmad, Stan Tomov, and Jack Dongarra. "Investigating the Benefit of FP16-Enabled Mixed-Precision Solvers for Symmetric Positive Definite Matrices Using GPUs". *Comput. Sci.* Springer Int. Publ., 2020, pp. 237–250.

Ahmad, Khalid, Hari Sundar, and Mary Hall. "Data-Driven Mixed Precision Sparse Matrix Vector Multiplication for GPUs". *ACM Trans. Archit. Code Optim.* 16.4 (2019). ISSN: 1544-3566.

Amestoy, Patrick et al. "Mixed Precision Low Rank Approximations and their Application to Block Low Rank LU Factorization". hal-03251738v2. 2021.

Baboulin, Marc et al. "Accelerating scientific computations with mixed precision algorithms". *Comput. Phys. Commun.* 180.12 (2009), pp. 2526–2533. ISSN: 0010-4655.

Bagnara, Roberto. "A unified proof for the convergence of Jacobi and Gauss–Seidel methods". *SIAM Review* 37.1 (1995), pp. 93–97.

Buttari, Alfredo et al. "Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance While Achieving 64-Bit Accuracy". *ACM Trans. Math. Softw.* 34.4 (2008). ISSN: 0098-3500.

Carson, Erin and Nicholas J. Higham. "Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions". *SIAM J. Sci. Comput.* 40.2 (2018), A817–A847.

Chan, SH, KK Phoon, and FH Lee. "A modified Jacobi preconditioner for solving ill-conditioned Biot's consolidation equations using symmetric quasi-minimal residual method". *Int. J. Numer. Anal. Methods. Geomech.* 25.10 (2001), pp. 1001–1025.

Clark, M.A. et al. "Solving lattice QCD systems of equations using mixed precision solvers on GPUs". *Comput. Phys. Commun.* 181.9 (2010), pp. 1517–1528. ISSN: 0010-4655.

Dalton, Steven et al. *Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations*. Version 0.5.0. 2014. URL: http://cusplibrary.github.io/.

Davis, Timothy A. and Yifan Hu. "The University of Florida Sparse Matrix Collection". *ACM Trans. Math. Softw.* 38.1 (2011). ISSN: 0098-3500.

Dolan, Elizabeth D and Jorge J Moré. "Benchmarking optimization software with performance profiles". *Math. Prog.* 91.2 (2002), pp. 201–213.

Gunawardena, Ananda D, SK Jain, and Larry Snyder. "Modified iterative methods for consistent linear systems". *Linear Algebra Appl.* 154 (1991), pp. 123–143.

HSL. A collection of Fortran codes for large scale scientific computation. http://www.hsl.rl.ac.uk/.

Langguth, Johannes et al. "Scalable Heterogeneous CPU-GPU Computations for Unstructured Tetrahedral Meshes". *IEEE Micro* 35.4 (2015), pp. 6–15.

Lee, F-H et al. "Performance of Jacobi preconditioning in Krylov subspace solution of finite element equations". *Int. J. Numer. Anal. Methods. Geomech.* 26.4 (2002), pp. 341–372.

Loe, Jennifer A. et al. "Experimental Evaluation of Multiprecision Strategies for GMRES on GPUs". *IEEE Int. Parallel Distrib. Process. Symp. Workshops*. 2021, pp. 469–478.

Martinez-Navarro, Hector et al. *Repository for modelling acute myocardial ischemia: simulation scripts and torso-heart mesh, University of Oxford*. 2019.

McCormick, Stephen F., Joseph Benzaken, and Rasmus Tamstorf. "Algebraic error analysis for mixed-precision multigrid solvers". *ArXiv* abs/2007.06614 (2020).

Ooi, Rise et al. "Effect of Mixed Precision Computing on H-Matrix Vector Multiplication in BEM Analysis". *Proc. Int. Conf. High Perf. Comput. Asia-Pacific Region*. Fukuoka, Japan: ACM, 2020, pp. 92–101.

Pratapa, Phanisri P, Phanish Suryanarayana, and John E Pask. "Anderson acceleration of the Jacobi iterative method: An efficient alternative to Krylov methods for large, sparse linear systems". *J. Comput. Physics* 306 (2016), pp. 43–54.

Saad, Yousef. *Iterative methods for sparse linear systems*. SIAM, 2003.

Turkel, Eli. "Preconditioning techniques in computational fluid dynamics". *Annu. Rev. Fluid Mech.* 31.1 (1999), pp. 385–416.

Vázquez, F., J. J. Fernández, and E. M. Garzón. "A New Approach for Sparse Matrix Vector Product on NVIDIA GPUs". *Concurr. Comput.* 23.8 (2011), pp. 815–826. ISSN: 1532-0626.

5 HISTORY OF CHANGES

Version	Author(s)	Date	Comment
0.1	Didem Unat	10.03.2023	Initial draft for internal review
0.1.1	Karl Fuerlinger	27.03.2023	Improved draft for coordinator review

Table 4 *Document History of Changes*