Appendix: Artifact Description/Artifact Evaluation

Artifact Description (AD)

I. Overview of Contributions and Artifacts

A. Paper's Main Contributions

- C_1 We propose a new shared memory approach for efficient and scalable symmetric SpMV based on the Divide-and-Conquer (DC) paradigm.
- C_2 We present a conflict-aware hybrid DC solution with a machine-learning model to further speed up symmetric SpMV.
- C_3 We evaluate our approach on X86 (Intel Xeon Gold 6258R) and ARM (HUAWEI Kunpeng 920) CPUs, showing significant performance improvement compared to the state-of-the-art.

B. Computational Artifacts

- A_1 The DC-based symmetric SpMV (this work) and the vanilla CSR-based SpMV (https://doi.org/10.5281/zenodo.13145136)
- A_2 SpMV and symmetric SpMV from Intel oneAPI Math Kernel Library (oneMKL) [1]
- A_3 Symmetric SpMV based on RACE [2] (see https://doi.org/10.5281/zenodo.13071391)

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1,C_2,C_3	Figures 2,9-12
A_2	C_3	Figures 9,11
A_3	C_3	Figures 9,11

II. Artifact Identification

A. Computational Artifact A_1

Relation To Contributions

A_1 implements the DC-based symmetric SpMV (this work) and the vanilla CSR-based SpMV (described as contributions C_1 and C_2). We compare A_1 with A_2 and A_3 for performance evaluations (described as contributions C_3).

Expected Results

The measured performance are reported in Gflops. Compared to SpMV and symmetric SpMV from Intel oneMKL (A_2) and RACE-based symmetric SpMV (A_3), our DC-based symmetric SpMV (A_1) should be faster for most matrices.

Expected Reproduction Time (in Minutes)

The expected computational time of this artifact on Intel Xeon Gold 6258R CPU is about 120 min. The expected computational time of this artifact on HUAWEI Kunpeng 920 CPU is about 150 min.

Artifact Setup (incl. Inputs)

Hardware: Intel Xeon Gold 6258R CPU@2.4GHz, and HUAWEI Kunpeng 920 CPU@2.6GHz.

Software: No other software packages are required to compile and run A_1.

Datasets / Inputs: 318 large numerically symmetric sparse matrices (with the row number $N_r > 50,000$ and the nonzero number $NNZ > 100,000$) from the SuiteSparse Matrix Collection[3].

Installation and Deployment: On X86, all computational artifacts are compiled in double precision with the -O3 option using Intel oneAPI 2022.1. On ARM, we use GCC 8.4.1. We use all the 28 cores on the X86 CPU, and all the 32 cores on the ARM CPU. We set the thread affinity for all tests and bind a software thread to an individual physical core (without hyperthreading). We run different SpMV implementations 100 times for each matrix and measure the averaged Gflops.

Artifact Execution

T_1: We use the 318 matrices as the input matrices, and randomly generate the corresponding input vector for each matrix. The dataset is then used as input for T_2 to perform SpMV.

T_2: We run the computational artifact A_1 100 times for each pair of input matrix and vector, and measure the averaged Gflops.

Artifact Analysis (incl. Outputs)

Based on the measured average Gflops, we can plot Figures 2 and 9-12.

B. Computational Artifact A_2

Relation To Contributions

A_2 (referred to as *oneMKL* and *oneMKL-SYM* in the paper) is used to compare with our work (A_1) for the performance evaluation on X86 (described as contribution C_3).

Expected Results

A_2 should be slower than our DC-based symmetric SpMV (A_1) on most of the 318 matrices.

Expected Reproduction Time (in Minutes)

The expected computational time of this artifact on Intel Xeon Gold 6258R CPU is about 120 min.

Artifact Setup (incl. Inputs)

Hardware: Intel Xeon Gold 6258R CPU@2.4GHz

Software: The required software package is Intel oneMKL 2022.1.

Datasets / Inputs: The same as the computational artifact A_1.

[1] https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html

[2] https://github.com/RRZE-HPC/RACE

[3] https://sparse.tamu.edu/

Installation and Deployment: The same as the computational artifact A_1.

Artifact Execution

The same as the computational artifact A_1.

Artifact Analysis (incl. Outputs)

Based on the measured average Gflops, we can plot Figures 9 and 11.

C. Computational Artifact A_3

Relation To Contributions

A_3 (referred to as *RACE* in the paper) is used to compare with our work (A_1) for the performance evaluation (described as contribution C_3).

Expected Results

A_3 should be slower than our DC-based symmetric SpMV (A_1) on most of the matrices.

Expected Reproduction Time (in Minutes)

The expected computational time of this artifact on Intel Xeon Gold 6258R CPU is about 80 min.

Artifact Setup (incl. Inputs)

Hardware: Intel Xeon Gold 6258R CPU@2.4GHz.

Software: All required software packages for RACE are available at the link (https://github.com/RRZE- HPC/RACE), and can be downloaded together with RACE.

Datasets / Inputs: The same as the computational artifact A_1.

Installation and Deployment: Please refer to the link (https://github.com/RRZE-HPC/RACE) to compile RACE, and refer to the computational artifact A_1 to deploy the experiments for this paper.

Artifact Execution

The same as the computational artifact A_1.

Artifact Analysis (incl. Outputs)

Based on the measured average Gflops, we can plot Figures 2, 9 and 11.

Artifact Evaluation (AE)

A. Computational Artifact A_1

Artifact Setup (incl. Inputs)

First download the zip package from the link (https://doi.org/10.5281/zenodo.13145136) and unzip it.

$unzip DCS-SpMV.zip

Then download the selected symmetric matrices from the SuiteSparse Matrix Collection using the script.

$python download_matrix.py

This process will take approximately 8 hours or longer. We need all these matrices to train the machine learning model. At the directory `DCS-SpMV/matrix`, we have downloaded two matrices which can be used directly for performance comparison.

Our zip package also contains SpMV and symmetric SpMV based on the Intel oneMKL library (i.e., the computational artifact A_2). If your system does not contain Intel oneAPI 2022.1, you need to manually download it and specify the compiler path in the `DCS-SpMV/Makefile`:

`line2: CC=/path/to/your/icpx`

After that, compile the source codes in the directory `DCS-SpMV/`.

$make

Then, you can get the executable files `profiling_SpMV` and `test_ML`.

Artifact Execution

According to the ML training workflow in Figure 8, we first profile each training matrix under different DC recursion numbers to find out the best-performing *K* and extract the matrix and conflicts features. Run the following script to profile.

$bash profile.sh

The results are saved in `performance_result.csv` and the feature values together with sub-optimal *K* are saved in `samples.csv`.

Then train the predictive model for DCH-SpMV.

$python training.py

After training, the predictive K_{hyb} of the matrices for the test set are saved in `predict.txt`. Then, you can run the script to test the prediction accuracy of K_{hyb}.

$bash test_ML.sh

After testing, the results are saved in `ml_result.csv`.

Artifact Analysis (incl. Outputs)

The performance results of the vanilla SpMV, the Intel oneMKL SpMV, the Intel oneMKL symmetric SpMV and our DCH-SpMV are saved in `performance_results.csv`. Based on the averaged Gflops, we can plot Figures 2 and 9-11. The performance loss due to mispredicted K_{hyb} are saved in `ml_result.csv`. Based on the performance comparisons, we can plot Figure 12.

B. Computational Artifact A_2

The computational artifact A_2 can be found in the same zip package of A_1. Please refer to A_1 for setup, execution and analysis.

C. Computational Artifact A_3

Please refer to the link (https://doi.org/10.5281/zenodo.13071391) to download, compile and run RACE.