Appendix: Artifact Description/Artifact Evaluation

Artifact Description (AD)

1 OVERVIEW OF CONTRIBUTIONS AND ARTIFACTS

We provide the following main components to help verify our experiments:

- our scheduling algorithms,
- the matrix data sets for our experiments,
- the baseline schedulers,
- our SpTRSV test suite, and scripts to run the algorithms,
- the output data from our experiments,
- the scripts used for the analysis of this data.

Our algorithms are all incorporated into our OneStopParallel scheduling toolbox, which is available on GitHub. However, for reproducing the experiments of the paper, we instead refer to a specific artifact repository [BMPS25] created for this purpose. This repository explicitly contains the version of the code used in our experiments, the data from our experiments, as well as scripts to download the data sets and to run the experiments.

1.1 Paper's Main Contributions

The paper presents a novel parallel scheduling algorithm for speeding up the task of sparse triangular system solving (SpTRSV). The main contributions of the paper are:

C_1: the new GrowLocal scheduling algorithm for efficient parallel SpTRSV execution;

C_2: the Funnel coarsening algorithm, and a theoretical proof that it preserves acyclicity;

C_3: our experiments, which show a significant speed-up of SpTRSV run time with our scheduler, compared to several baselines.

Besides this, our experimental results also confirm several further observations, which can be understood as parts of C_3:

$C_{3.1}$: our improvements are consistent over several different architectures;

$C_{3.2}$: our algorithm also scales reasonably to a larger number of cores;

$C_{3.3}$: a block decomposition technique that decreases the scheduling time.

1.2 Computational Artifacts

The concrete list of artifacts and their location is summarized in Table 1. The artifacts are currently in the GitHub folder [BMPS25]. We plan to archive them under a single DOI upon publication.

Note from Table 1 that the artifacts are interdependent. Artifacts A_1–A_3 are the main components required for our experiments: our algorithms, the data sets, and the baselines. Artifacts A_4 contains the scripts that run our experiment; this relies on A_1–A_3. Artifacts A_5–A_9 are data analysis scripts that use to the output data from A_4 to produce the tables and figures in our table. As Artifact A_{10}, we also include the data we obtained from the experiments, which can also be used as backup to reproduce our results with the same scripts as in A_5–A_9.

The file 'README.md' in this repository guides the user through the process of running Artifacts A_1–A_9 in order.

2 ARTIFACT IDENTIFICATION

2.1 Computational Artifact A_1

Relation To Contributions

This artifact contains our main scheduling algorithms from the paper: GrowLocal, the Funnel coarsener, and the reordering technique.

Our scheduling algorithms provided as part of our OneStopParallel toolbox in the folder 'one-stop-parallel'. OneStopParallel is a versatile tool with many algorithms; the codes that are relevant for this work are as follows:

- the code of the GrowLocal scheduling can be found in `source/scheduler/GreedySchedulers/SMGreedyBspGrowLocalAutoCoresParallel.cpp`;
- the code of the Funnel coarsener can be found in the file `source/scheduler/GreedySchedulers/SMFunOriGrowlv2.cpp`;
- the schedule reordering step is implemented in `source/scheduler/SchedulePermutations/ScheduleNodePermuter.cpp`.

Expected Results

Upon compilation, the algorithms can be used in the test suite of A_4, to produce the data for A_5–A_9.

Expected Reproduction Time (in Minutes)

The compilation of the OneStopParallel toolbox took 5 to 15 minutes in our case, but may vary depending on architecture.

Artifact Setup (incl. Inputs)

Hardware. OneStopParallel can be compiled on various platforms.

For our main experiments, we used the following architectures:

- Intel Xeon Gold 6238T processor (x86), with 192 GB memory and theoretical peak memory throughput of 140.8 GB/s and 22 cores on a single socket; kernel version 5.14.0; GCC version 11.5.0;
- AMD EPYC 7763 processor (x86), with 1024 GB memory and theoretical peak memory throughput of 204.8 GB/s and 64 cores on a single socket; kernel version 5.15.0; GCC version 11.4.0;
- Huawei Kunpeng 920-4826 (Hi1620) processor (ARM), with 512 GB memory and theoretical peak memory throughput of 187.7 GB/s and 48 cores on a single socket; kernel version 5.15.0; GCC version 11.4.0.

Software. The software dependencies required by OneStopParallel, as well as the commands to install them on Ubuntu, are as follows (listed in the pre-requisites section of the main script):

Table 1: List of main artifacts for the paper.

Artifact ID	Contributions Supported	Related Paper Elements
A_1: Code of our schedulers	C_1, C_2, C_3	GrowLocal, Funnel+GL algorithms
A_2: Matrix data sets	C_3	The 5 data sets from Section 6.2
A_3: Guide to run baseline algorithms	C_3	SpMP, HDagg baselines
A_4: SpTRSV test suite and scripts	C_3	used for the experiments
A_5: Main SpTRSV speed-ups	C_3	Tables 7.1, 7.2, 7.6 Figures 1.2, 7.1
A_6: Effect of reordering on speed-ups	C_3	Table 7.3
A_7: Consistent speed-ups over architectures	$C_{3.1}$	Table 7.4
A_8: Scaling to more cores	$C_{3.2}$	Table 7.5, Figure 7.2
A_9: Block decomposition: reduced scheduling time	$C_{3.3}$	Table 7.7
A_{10}: Our result data & analysis scripts	C_3, $C_{3.1}$, $C_{3.2}$, $C_{3.3}$	To confirm A_5–A_9 with the original data from our experiments

- cmake: `sudo apt-get install cmake`
- OMP: `sudo apt-get install libomp-dev`
- Boost library [Boo15] and its dependencies: `sudo apt-get install build-essential g++ python-dev autotools-dev libicu-dev libbz2-dev libboost-all-dev`
- Eigen library: `sudo apt-get install libeigen3-dev`

Datasets / Inputs. No inputs required.

Installation and Deployment. OneStopParallel can be compiled easily with cmake. The concrete commands are also provided in the separate 'bash build_one-stop-parallel.sh' script.

Artifact Execution

Upon compilation, OneStopParallel is built, and the algorithms can be used for Artifact A_4.

Artifact Analysis (incl. Outputs)

Successful compilation.

2.2 Computational Artifact A_2

Relation To Contributions

We use 5 different matrix sets as our datasets, as explained in Section 6.2 of the paper. We provide a script that allows to download/generate these data sets:

- the SuiteSparse data set is downloaded by the script from https://sparse.tamu.edu/, cf. [DH11];
- the METIS dataset is converted from SuiteSparse with sympiler, as in the reference paper introducing HDagg;
- the iChol dataset is converted from SuiteSparse with the help of Eigen;
- the Erdős-Rényi and Narrow Bandwidth datasets are generated randomly.

Note that for the random datasets, the matrices obtained with this script are not exactly identical to those used in our experiments, due to the randomization; however, their general properties and behavior should be almost identical.

Expected Results

Upon compilation, the matrices in the data sets can be used for A_4, to produce baseline data for A_5–A_9.

Expected Reproduction Time (in Minutes)

The script to download the SuiteSparse data set and convert/generate the remaining data sets took 90-120 minutes for us, but can be dependent on the architecture and internet connection.

Artifact Setup (incl. Inputs)

Hardware. Same as in Artifact A_1. We used the Intel x86 processor to generate the data sets, but the generation should also be possible on the other two architectures.

Software. The artifact requires cmake, OMP and Eigen, similarly to Artifact A_1 before. An additional dependency (and Ubuntu installation command) is as follows:

- METIS library: `sudo apt-get install libmetis-dev`

A clean version of sympiler is required to obtain the METIS data set. This is set up by the separate 'downloadSympilerForMatrixConversion.sh' script. OneStopParallel is required to generate the Erdős-Rényi, Narrow Bandwidth, and iChol data sets. OneStopParallel can be build with the bash script 'build_one-stop-parallel.sh".

Datasets / Inputs. No inputs required.

Installation and Deployment. One needs to run the following three scripts in this order:

- downloadSympilerForMatrixConversion.sh

- build_one-stop-parallel.sh
- downloadDatasets.sh
- generateGraphStatistics.sh

Artifact Execution

The scripts above need to be run in this order.

The script downloadSympilerForMatrixConversion.sh downloads the Sympiler framework of Zarebavani *et al.* [ZCL$^+$22], applies a patch to it, and then builds it. This creates the METIS dataset that was previously used in the paper of Zarebavani *et al.* [ZCL$^+$22] for their experiments. The patch is required to ensure that the modified sympiler does nothing else besides permuting the input matrices with the METIS partitioner, outputs the result in the appropriate format, and then exits the program.

The main script downloadDatasets.sh then creates the 5 data sets. SuiteSparse [DH11] is downloaded from its online repository. METIS is converted from this with the sympiler above, and iChol is converted from this using Eigen. The two random datasets are generated with OneStopParallel.

The final script generateGraphStatistics.sh analyzes the matrices and provides some basics statistics for it (e.g. number of columns, non-zeros, etc.). We need to generate these because the random matrices may differ every time we generate them.

Artifact Analysis (incl. Outputs)

Upon compilation, the 5 data sets are successfully generated for Artifact A_4.

2.3 Computational Artifact A_3

Relation To Contributions

HDagg [ZCL$^+$22] and SpMP [PSSD14] are the two main baselines for our SpTRSV schedulers. We run these algorithms through the Sympiler framework of Zarebavani *et al.* [ZCL$^+$22], which contains the original implementation of HDagg, and also provides an interface to run the original implementation of SpMP.

Here we provide a readme on how to download and set up sympiler. We also apply a brief patch to sympiler to ensure that the configuration of their framework and our own is identical (number of runs, hot starts, etc.).

Expected Results

Upon compilation, sympiler can be used in the test suite of A_4, to produce baseline data for A_5–A_9.

Expected Reproduction Time (in Minutes)

The compilation of sympiler and the application of the patch should take $5 - 15$ minutes.

Artifact Setup (incl. Inputs)

Hardware. Sympiler can be compiled any of the three architectures discussed in the Hardware section of Artifact A_1; however, the process is slightly different.

On Intel or AMD x86, one should use the script ‘downloadSpMP-SympilerForExperiments.sh’. This also includes the SpMP baseline in sympiler.

On the other hand, on ARM, one should use the script ‘download-ARMSympilerForExperiments.sh’. This leaves SpMP out (since the implementation of the SpMP baseline is x86-specific), and builds sympiler with only HDagg on ARM.

Software. The software dependencies are OMP and METIS, which are discussed in Artifacts A_1 and A_2, respectively.

Datasets / Inputs. No inputs required.

Installation and Deployment. The steps are summarized in the script ‘downloadSpMPSympilerForExperiments.sh’ or ‘downloadARM-SympilerForExperiments.sh’, depending on architecture.

Artifact Execution

Setting up these baseline experiments consists of multiple steps:

- download and configure the Sympiler framework of Zarebavani *et al.* from https://github.com/sympiler/aggregation;
- on x86: download, patch and compile the SpMP baseline of Park *et al.* from https://github.com/IntelLabs/SpMP, and configure Sympiler to use this implementation;
- apply our patch files to Sympiler in order to align the experimental setup to ours;
- build Sympiler.

Upon compilation, the baselines SpMP and HDagg can be used for Artifact A_4.

Artifact Analysis (incl. Outputs)

Successful compilation of the adapted sympiler framework. The baselines HDagg and SpMP (if applicable) can then be used for Artifact A_4.

2.4 Computational Artifact A_4

Relation To Contributions

A test suite to evaluate the our SpTRSV schedulers is contained within OneStopParallel, in the folder test_suite. The test suite to evaluate the baselines is within Sympiler. The serial running times in the two test suite are essentially identical, which confirms the validity of this setup.

We also provide a script that allows to run the experiments conveniently. We provide 6 scripts altogether, for our algorithms and the baselines on the 3 considered architectures.

Expected Results

Upon running, the script produces the data files that can be used in the data analysis for the baselines A_5–A_9.

Expected Reproduction Time (in Minutes)

Running Sympiler on the SuiteSparse data sets can take up to around 1-2 days. The data sets METIS, iChol, Erdős–Rényi should finish in less time, up to around 1 day per data set. Running the scripts on the Narrow Bandwidth dataset is typically a matter of hours only. Hence, running our bash script ’run_sympiler_experiments_Intel.sh’, that goes through all the data sets, can take up to around 4-5 days. The bash script can be changed to run only single data sets and break the simulation up into shorter runs. For that a targeted data set can be specified in the *args* list (lines 19 to 25).

We note that this large running time is mostly due to the baseline algorithms. In order to modify the Sympiler baseline as little as possible, we also run the scheduling algorithm separately for each SpTRSV experiment (apart from those ignored for the warm start). This results in a much higher runtime altogether when running the baseline experiment 100 times.

Running our algorithms on the SuiteSparse data set only takes 2-3 hours typically.

We note that the output data from the baseline experiments, recorded on our machines, is also available with all other data in the folder 'our_result_data_and_analysis', subfolder 'SpTrSV_Data'. As such, if one wants to avoid the compilation of this external code, in Artifact A_{10}, pre-run data can also be used to verify our results. As such, if the running time of multiple days is not desired, then it is also possible to 1) use the data in Artifact A_{10} to verify artifacts A_5–A_9, and 2) only run the experiments in Artifact A_4 with our algorithms (but not the baselines), or only on a subset of the matrices, and compare this to the files in Artifact A_{10} to confirm that the numbers are very similar.

Artifact Setup (incl. Inputs)

Hardware. As discussed in Section 6.3 of the paper, and in the Hardware section of Artifact A_1, we run our experiments on three different computing architectures: Intel x86, AMD epyc x86 and ARM.

Our main experiments were conducted with 22 cores. As such, in order to run the experiments corresponding to the Intel x86 and ARM machines, an architecture is needed that has at least **22 cores on a single socket**.

The experiments on AMD epyc also check the scaling of our scheduler to a higher number of cores (up to 64), but still without NUMA effects. in order to run the experiments corresponding to AMD epyc, an architecture is needed that has at least **64 cores on a single socket**.

Software. The components from artifacts A_1–A_3 are required.

Eigen, Boost, OMP, METIS and cmake need to be installed, as in previous artifacts.

Datasets / Inputs. The input datasets from artifact A_2 are required.

Installation and Deployment. The experiments for our algorithms on the 3 respective architectures can be run with the following scripts from the 'one-stop-parallel' folder:

- 'run_sc25_experiment_Intel.sh';
- 'run_sc25_experiment_AMD.sh';
- 'run_sc25_experiment_ARM.sh'.

The baselines can be run on the 3 respective architectures with the following scripts from the main folder:

- 'run_sympiler_experiments_Intel.sh';
- 'run_sympiler_experiments_AMD.sh';
- 'run_sympiler_experiments_ARM.sh'.

Please make sure to only use the script **corresponding to the appropriate architecture**.

Artifact Execution

The six scripts can be run in any desired order; the output data is collected in separate files.

Each separate script only sets the appropriate OMP flags and the number of cores, and then calls the appropriate test suite ('Hdagg_SpTRSV' for sympiler, or a test suite execution file from OneStopParallel).

The scripts also move the files to the appropriate folder in the end.

The number of experiments in the specific scripts differ. The scripts for ARM and AMD epyc only consider the main data set (SuiteSparse), except for the scalability study (Artifact A_8), which is executed for our algorithms on AMD epyc. In contrast, for Intel x86, all 5 data sets are run. There are even more experiments in the script four our algorithms on SuiteSparse: this is where the different ablation studies are executed (Artifacts A_6, A_9).

Artifact Analysis (incl. Outputs)

The results are output into the .csv files. The running times for each algorithm and matrix should be comparable to the data from our experiments, which are available in the 'our_result_data_and_analysis' folder in the 'SpTrSV_Data' subfolder.

The output data from the experimetns is automatically inserted into the appropriate order for the data processing in Artifacts A_5–A_9.

2.5 Computational Artifact A_5

Relation To Contributions

The main experiments demonstrate the SpTRSV speed-ups over the baselines. Given the output from Artifact A_4, the Jupyter notebook named 'Data_Analysis_SC_paper_clean' processes these results to produce Tables 7.1, 7.2, 7.6 and Figures 1.2, 7.1. The concrete correspondence between the notebook cells and the tables/figures is outlined in the readme file in the 'our_result_data_and_analysis' folder, and also discussed below.

We note that in the raw data in the notebooks, the data sets and algorithms have slightly different names than in the paper[1]:

- for the data sets, florida (SuiteSparse), florida_metis (METIS), florida_Pchol (iChol), er (Erdős_Rényi), rb (Narrow Bandwidth);
- for the algorithms, HDAGG_BIN (HDagg), SpMP (SpMP), SMGreedyBspGrowLocalAutoCoresParallel (GrowLocal), SMFunOriGrowlv2 (Funnel+GL).

Expected Results

Upon running, the notebook should produce tables and figures similar to Tables 7.1, 7.2, 7.6 and Figures 1.2, 7.1.

Expected Reproduction Time (in Minutes)

Running the pythons scripts in the notebook should take at most 1-5 minutes.

[1]The same notational difference holds for Artifacts A_6–A_{10}.

Artifact Setup (incl. Inputs)

Hardware. We used an Intel Xeon Gold 6238T processor (x86), with 192 GB memory and theoretical peak memory throughput of 140.8 GB/s.

Software. We use the following Python version and package versions to run the Jupyter notebook:

- Python version: 3.11.0
- matplotlib version: 3.10.0
- pandas version: 2.2.3
- numpy version: 2.2.1
- scipy version: 1.15.0
- scikit-learn version: 1.6.1
- seaborn version: 0.13.2
- jupyter notebook:
  - IPython : 8.31.0
  - ipykernel : 6.29.5
  - jupyter_client : 8.6.3
  - jupyter_core : 5.7.2
  - traitlets : 5.14.3

Datasets / Inputs. The input datasets from artifact A_4 are required.

Installation and Deployment. The Jupyter notebook can be run with various tools, e.g. VSCode.

Artifact Execution

One should simply run all the cells in the Jupyter notebook in order.

Artifact Analysis (incl. Outputs)

The results are output into Jupyter notebook itself. We note that the beginning of our different notebooks in A_5–A_9 are essentially identical, just loading data from different folders. The end of the notebooks then produce the tables and figures from the tables.

Cells 38-42 process our main SpTRSV speed-ups for each of the 5 data sets. The "Geommean_serial" column provides the numbers in Table 7.1 for each data set. The "Geommean_supersteps_relative_to_Wavefront" column provides the numbers in Table 7.2 for each data set. The other two columns are only the relative speed-ups between the, i.e. the ratios of the values in the other two columns. We note that the numbers for Narrow Bandwidth are slightly different from those in the paper, since due to an error, a single instance was left out for the SpMP and HDagg runs; however, the difference in the final numbers is very small.

Cell 43 processes the amortization threshold on the florida (SuiteSparse) data set, thus producing the numbers for Table 7.6 in the q25, median_profitability and q75 columns for each algorithm. Note that some of the values slightly differ from the paper, but this is only random noise: we ran these experiments again for Table 7.7 (see later), and used the same value in both tables to avoid confusion.

Cell 44 produces the violin plot that appears as Figure 1.2 in the paper.

Cell 46 produces a plot of the scheduling time that is currently included in the supplementary material.

Cell 50 produces the performance plot that appears as Figure 7.1 in the paper. We note that due to a small fix in the Funnel+GL algorithm, one of the curves slightly differs from the variant in the paper.

2.6 Computational Artifact A_6

Relation To Contributions

The next experiment demonstrates the effect of the reordering technique on the SpTRSV speed-up. Given the output from Artifact A_4, the Jupyter notebook named 'Data_Analysis_paper_permutation_clean' processes these results to produce the numbers in Table 7.3. The concrete correspondence between the notebook cells and the table is outlined in the readme file in the 'our_result_data_and_analysis' folder, and also discussed below.

Expected Results

Upon running, the notebook should produce numbers similar to Table 7.3.

Expected Reproduction Time (in Minutes)

Running the pythons scripts in the notebook should take at most 1–2 minutes.

Artifact Setup (incl. Inputs)

Hardware. Same as in Artifact A_5.

Software. Same as in Artifact A_5.

Datasets / Inputs. Same as in Artifact A_5.

Installation and Deployment. Same as in Artifact A_5.

Artifact Execution

One can simply run all the cells in the Jupyter notebook.

Artifact Analysis (incl. Outputs)

The results are output into Jupyter notebook itself.

The data here can be used to reproduce Table 7.7 in the paper. Cell 33 produces a table where the "median_profitability" column gives the data in "Amort. Threshold" column of Table 7.7. Cell 34 produces a table where the columns "GM_MulThr_sched_time_speedup" "GM_MulThr_FLOP_decrease" and "GM_MulThr_superstep_increase", respectively, give the values in the "Sched. Time", "Flops/s" and "Supersteps" columns of Table 7.7.

2.7 Computational Artifact A_7

Relation To Contributions

The next experiment studies the SpTRSV speed-up over several architectures. Given the output from Artifact A_4, the Jupyter notebook named 'Data_Analysis_paper_arch_clean' processes these results to produce the numbers in Table 7.4. The concrete correspondence between the notebook cells and the table is outlined in the readme file in the 'our_result_data_and_analysis' folder, and also discussed below.

Expected Results

Upon running, the notebook should produce numbers similar to Table 7.4.

Expected Reproduction Time (in Minutes)

Running the pythons scripts in the notebook should take at most 1–2 minutes.

Artifact Setup (incl. Inputs)

Hardware. Same as in Artifact A_5.

Software. Same as in Artifact A_5.

Datasets / Inputs. Same as in Artifact A_5.

Installation and Deployment. Same as in Artifact A_5.

Artifact Execution

One should simply run all the cells in the Jupyter notebook in order.

Artifact Analysis (incl. Outputs)

The results are output into Jupyter notebook itself.

Cell 31 shows all the values for Table 7.3, in the appropriate order of the data sets. LOOP_PROCESSORS corresponds to the "Reordering" column, and NO_PERMUTE corresponds to the "No Reordering" column.

2.8 Computational Artifact A_8

Relation To Contributions

The next experiment studies how our speed-up scales with the number of cores. Given the output from Artifact A_4, the Jupyter notebook named 'Data_Analysis_paper_scaling_clean' processes these results to produce the numbers in Table 7.5. and Figure 7.2. The concrete correspondence between the notebook cells and the table is outlined in the readme file in the 'our_result_data_and_analysis' folder, and also discussed below.

Expected Results

Upon running, the notebook should produce numbers similar to Table 7.5., and a figure similar to Figure 7.2.

Expected Reproduction Time (in Minutes)

Running the pythons scripts in the notebook should take at most 1–2 minutes.

Artifact Setup (incl. Inputs)

Hardware. For the notebook, the same as in Artifact A_5.

Recall that for the experiments producing this data, we used an AMD EPYC 7763 processor (x86), with 1024 GB memory and theoretical peak memory throughput of 204.8 GB/s, because this machine had 64 cores available on a single socket. To verify our experiments, we believe it is crucial to use a machine with up to 64 cores **on a single socket**, otherwise we can critical interference form NUMA effects.

Software. Same as in Artifact A_5.

Datasets / Inputs. Same as in Artifact A_5.

Installation and Deployment. Same as in Artifact A_5.

Artifact Execution

One should simply run all the cells in the Jupyter notebook in order.

Artifact Analysis (incl. Outputs)

The results are output into Jupyter notebook itself.

In Cell 30, the "Geomean" column of the table shows the speed-ups for each specific architecture and algorithm in Table 7.4.

2.9 Computational Artifact A_9

Relation To Contributions

The next experiment studies the block decomposition technique to speed up the scheduling process. Given the output from Artifact A_4, the Jupyter notebook named 'Data_Analysis_SC_paper_scheduling _thread_clean' processes these results to produce the numbers in Table 7.7. The concrete correspondence between the notebook cells and the table is outlined in the readme file in the 'our_result_data_and_analysis' folder, and also discussed below.

Expected Results

Upon running, the notebook should produce numbers similar to Table 7.7.

Expected Reproduction Time (in Minutes)

Running the pythons scripts in the notebook should take at most 1–2 minutes.

Artifact Setup (incl. Inputs)

Hardware. Same as in Artifact A_5.

Software. Same as in Artifact A_5.

Datasets / Inputs. Same as in Artifact A_5.

Installation and Deployment. Same as in Artifact A_5.

Artifact Execution

One should simply run all the cells in the Jupyter notebook in order.

Artifact Analysis (incl. Outputs)

The results are output into Jupyter notebook itself.

Cell 34 shows the speed-ups of GrowLocal for different numbers of cores (note that Table 7.5 only shows some selected rows). Cell 35 generates the plot for Figure 7.2; this looks slightly different from the paper due to some last minute data fixes, but its essential characteristics are the same.

2.10 Computational Artifact A_{10}

Relation To Contributions

As mentioned before, for completeness, we also upload the result data files from our experiments to the folder 'our_result_data_and_analysis', subfolder 'SpTrSV_Data'. We name this as a separate artifact because it is not strictly related to any of A_1–A_9, but it allows an easy verification of our data processing in the Artifacts A_5–A_9 in case of any obstacles to running Artifacts A_1–A_4. A copy of each Jupyter notebook from A_5–A_9 is also provided in the 'our_result_data_and_analysis' folder for convenience.

Expected Results

Upon running, the notebook should produce the results of Artifacts A_5–A_9.

Expected Reproduction Time (in Minutes)

Running the pythons scripts in the notebook should take at most 1–2 minutes for each of Artifacts A_5–A_9.

Artifact Setup (incl. Inputs)

Hardware. Same as in Artifact A_5.

Software. Same as in Artifact A_5.

Datasets / Inputs. Data from folder 'our_result_data_and_analysis'.

Installation and Deployment. Same as in Artifact A_5.

Artifact Execution

The folder contains copies of all the Jupyter notebooks from Artifacts A_5–A_9. One should simply run all the cells in the Jupyter notebook in order.

Artifact Analysis (incl. Outputs)

The results are identical to those of Artifacts A_5–A_9.

REFERENCES

[BMPS25] Toni Böhnlein, Christos Matzoros, Pál András Papp, and Raphael S. Steiner. Artifact Repository. https://github.com/Algebraic-Programming/Artifacts/tree/master/SC_2025_Efficient_Parallel_Scheduling_Sparse_Triangular_Solvers, 2025.

[Boo15] Boost. Boost C++ Libraries. http://www.boost.org/, 2015. Last accessed 08-04-2025.

[DH11] Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1–25, 2011.

[PSSD14] Jongsoo Park, Mikhail Smelyanskiy, Narayanan Sundaram, and Pradeep Dubey. Sparsifying synchronization for high-performance shared-memory sparse triangular solver. In *Supercomputing: 29th International Conference, ISC 2014, Leipzig, Germany, June 22-26, 2014. Proceedings 29*, pages 124–140. Springer, 2014.

[ZCL+22] Behrooz Zarebavani, Kazem Cheshmi, Bangtian Liu, Michelle Mills Strout, and Maryam Mehri Dehnavi. HDagg: hybrid aggregation of loop-carried dependence iterations in sparse matrix computations. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1217–1227. IEEE, 2022.