Replicated Computational Results (RCR) Report for "Adaptive Precision Block-Jacobi for High Performance Preconditioning in the Ginkgo Linear Algebra Software"

S. V. Osborn

June 11, 2021

Association for Computing Machinery Transactions on Mathematical Software

Replicated Computational Results (RCR) Report for "Adaptive Precision Block-Jacobi for High Performance Preconditioning in the Ginkgo Linear Algebra Software"

SARAH OSBORN, Lawrence Livermore National Laboratory

The article by Flegar et al. titled "Adaptive Precision Block-Jacobi for High Performance Preconditioning in the Ginkgo Linear Algebra Software" presents a novel, practical implementation of an adaptive precision block-Jacobi preconditioner. Performance results using state-of-the-art GPU architectures for the block-Jacobi preconditioner generation and application demonstrate the practical usability of the method, compared to a traditional full-precision block-Jacobi preconditioner. A production-ready implementation is provided in the Ginkgo numerical linear algebra library.

In this report, the Ginkgo library is reinstalled and performance results are generated to perform a comparison to the original results when using Ginkgo's Conjugate Gradient solver with either the full or the adaptive precision block-Jacobi preconditioner for a suite of test problems on an NVIDIA GPU accelerator. After completing this process, the published results are deemed reproducible.

CCS Concepts: • **General and reference → Verification**; • **Mathematics of computing → Mathematical software performance**;

Additional Key Words and Phrases: Krylov subspace methods, block-Jacobi preconditioning, GPU, adaptive precision, replicated computational results

ACM Reference format:
Sarah Osborn. 2021. Replicated Computational Results (RCR) Report for "Adaptive Precision Block-Jacobi for High Performance Preconditioning in the Ginkgo Linear Algebra Software". *ACM Trans. Math. Softw.* 47, 2, Article 15 (March 2021), 4 pages.
https://doi.org/10.1145/3446000

1 INTRODUCTION

In the work of Flegar et al. [2], a practical implementation of a novel adaptive precision block-Jacobi preconditioner is introduced. In particular, the authors present a heavily tuned

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Author's address: S. Osborn, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P. O. Box 808, L-561, Livermore, CA 94551; email: osborn9@llnl.gov.

2021. 0098-3500/2021/03-ART15 $15.00
https://doi.org/10.1145/3446000

GPU implementation of the adaptive precision block-Jacobi preconditioner within the Ginkgo numerical linear algebra library. The performance of the methodology and implementation is demonstrated using the proposed preconditioning scheme within Ginkgo's high-performance Conjugate Gradient (CG) implementation on an NVIDIA Volta GPU.

In this report, we replicate a subset of the computational results presented by Flegar et al. [2]. The focus is generating results from Figure 9 to evaluate the performance of using Ginkgo's CG solver integrated with either the full or the adaptive precision block-Jacobi preconditioner applied to a variety of test cases.

The main steps are as follows to replicate the experimental results:

(1) Install the ssget tool and prefetch test matrices from the SuiteSparse collection.
(2) Download and build Ginkgo.
(3) Prepare the experiment scripts.
(4) Run the experiments.
(5) Publish the experiments to a git repository and use the Ginkgo Performance explorer (an interactive webtool) to generate the plots.

Although the results from the article are benchmarked using the Summit supercomputer at Oak Ridge National Laboratory, the replication results are generated using Lassen at Lawrence Livermore National Laboratory. Both systems are composed of the same IBM Power9 CPUs and NVIDIA Tesla Volta V100 GPU accelerators; however, there are differences in the ratio of CPU to GPU—Summit has six GPUs for every pair of Power9 chips compared to Sierra's four GPUs per pair of CPUs. For both sets of experiments ([2] and the replication effort), a single NVIDIA V100 GPU accelerator is used so the results will be comparable.

2 REPLICATION OF COMPUTATIONAL RESULTS OF THE ARTICLE

A detailed description of the configuration used to generate the performance results has been provided to the reviewer by the authors. These instructions are openly available in a markdown document [1], which can be accessed at https://github.com/ginkgo-project/ginkgo/blob/2019toms-adaptive-bj-solver/Reproduce_Experiments.md. All software components and performance evaluation tools are openly available via GitHub repositories. It is assumed that the author-provided instructions are followed exactly unless noted otherwise. Only a summary of the specific instructions and commands will be outlined in the following for the sake of brevity, as full details can be found on GitHub [1].

2.1 Software Download and Installation

2.1.1 Download Test Matrices. First, the ssget tool is cloned from the git repository (https://github.com/ginkgo-project/ssget) to facilitate downloading the test matrices from the SuiteSparse matrix collection. A bash script is provided to pre-download the test matrices using ssget. The downloaded test matrices are stored in `/p/gpfs1/<username>`, the recommended location for parallel file space on Lassen.

2.1.2 Download and Build Ginkgo. The source code is downloaded from the git repository (https://github.com/ginkgo-project/ginkgo.git) using the 2019toms-adaptive-bj-solver branch. CMake (version 3.14.5) is used to set up the build system, where it is specified to build optimized CUDA versions of the kernels using CUDA version 9.2.148 as in the work of Flegar et al. [2]. Due to different available versions of GCC on Lassen, GCC is changed to use version 7.3.1 (instead of version 6.4.0).

Building the Gingko project is straightforward, whereas making sure to use a compute node for the `make -j10` step as a CUDA bug leads to a slow compilation process. Once the project is compiled, all 79 unit tests passed after running `make test`.

2.2 Replicating the Experiments

First, two files are created to launch the experiments, following Step 3 from GitHub [1]. Note that due to slight differences in Lassen and Summit, some BSUB parameters must be altered in the benchmark_ginkgo.lsf file. The following lines are added for Lassen runs.

```
#BSUB -G account #Replace #BSUB -P ${project^^}
#BSUB -q pbatch #Specifies the name of the queue to use
```

In the benchmark_one_node.sh file, `export SYSTEM_NAME=V100_lassen` is used to differentiate the results, as this variable is used for the folder name to store the experiment output files. To launch the experiments, the command from Step 4 on GitHub [1] to benchmark all matrices and run 20 benchmarks in parallel is used. By default, one digit of the precondition is preserved. The experiments are run a second time where two digits of the preconditioner are preserved, following Step 6 from GitHub [1]. Recall that one NVIDIA V100 GPU accelerator is employed for the experiments.

2.3 Evaluation of Replicated Results

For each test matrix, a json file is generated that contains the timing and convergence results for the CG solver without preconditioning, with a standard (full-precision) block-Jacobi preconditioner, and with the adaptive precision block-Jacobi preconditioner.

Using Ginkgo's open source plotting tool, Ginkgo Performance Explorer (https://ginkgo-project.github.io/gpe/), similar figures to Figure 9 in the work of Flegar et al. [2] can be easily reproduced. The Ginkgo software project has an open source git repository that contains performance benchmarking data (https://github.com/ginkgo-project/ginkgo-data). In particular, the ginkgo-data repository contains the json files for the benchmarking experiments of Flegar et al. [2].

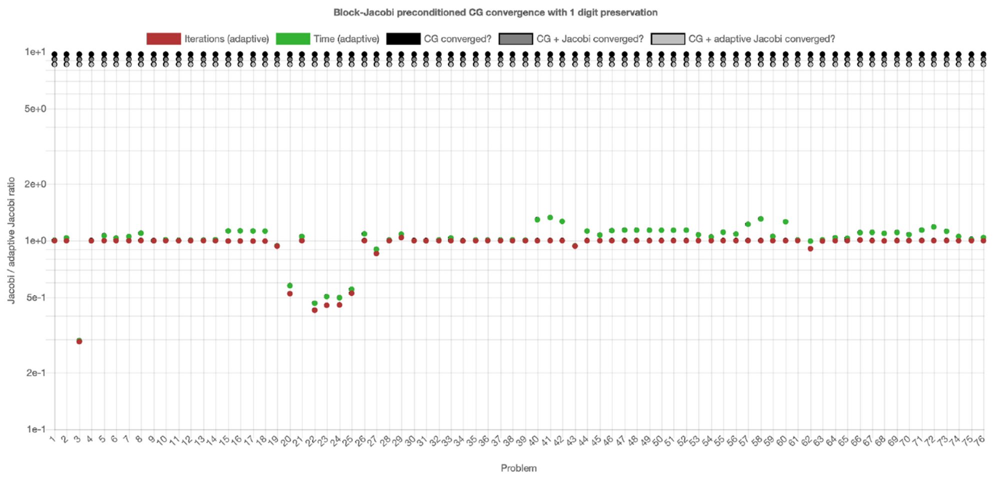
To evaluate the performance of our experiments on Lassen, first the results are published to a fork of the ginkgo-data repository (https://github.com/ginkgo-project/ginkgo-data/tree/2019toms-adaptive-bj) following Step 5 from GitHub [1]. Note that for this set of results, `build-list` should be altered so that `SYSTEM_NAME` is used in line 14 in place of `A100`. Once these results are loaded into Ginkgo Performance Explorer, the figures like those in Figure 9 in the work of Flegar et al. [2] are generated and shown in Figure 1.

Upon inspection of these results generated on Lassen, we note that the CG iteration counts are the same as those of Flegar et al. [2]. This comparison is conducted using the performance data from the 2019toms-adaptive-bj branch of the ginkgo-data repository, since only the relative number of iterations between the two preconditioners are reported in the work.

For the runtimes of the CG solver, the timing results are comparable. There are some fluctuations for some of the test cases of the relative timing results, yet when improvements are observed the speedup is between 10% and 30%, which is consistent with the observations of the article. In our estimation, the timing results are close enough to consider the results of Flegar et al. [2] replicated.

3 CONCLUDING REMARKS

By following the comprehensive instructions provided by Flegar et al. [2], all of the software components used in the results section were reinstalled. New performance results for comparing Ginkgo's CG solver with the full-precision block-Jacobi preconditioner to the adaptive precision block-Jacobi preconditioner were generated and compared to the original results. The necessary software to replicate the results is freely and openly available. Additionally, the availability



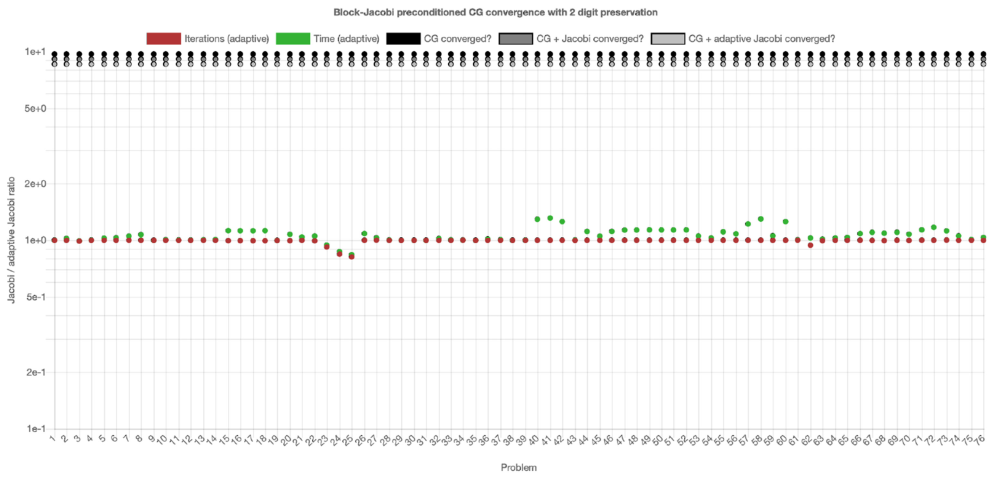


Fig. 1. Replication of Figure 9 from Flegar et al. [2] using the Lassen system where the adaptive precision preserves one digit (top) or two digits (bottom) of the full-precision block-Jacobi preconditioner. When compared to Figure 9, the results indicate that the iteration count is the same as the original, and similar speedups are achieved for certain test cases when using the CG solver enhanced with the adaptive precision block-Jacobi preconditioner compared to the CG solver with the full-precision variant of block-Jacobi.

of Ginkgo Performance Explorer, an interactive webtool for analyzing and plotting, made the process of analyzing the results very convenient. After completing this process, the published results are deemed replicable by the reviewer.

REFERENCES

[1] GitHub. n.d. Instructions to Reproduce Experiments. Retrieved October 23, 2020 from https://github.com/ginkgo-project/ginkgo/blob/2019toms-adaptive-bj-solver/Reproduce_Experiments.md.

[2] Goran Flegar, Hartwig Anzt, Terry Cojean, and Enrique S. Quintana-Orti. 2020. Adaptive precision block-Jacobi for high performance preconditioning in the Ginkgo linear algebra software. *ACM Transactions on Mathematical Software* 1, 1 (Aug. 2020), Article 1, 27 pages.

Received November 2020; accepted December 2020