Critique of "Planetary Normal Mode Computation: Parallel Algorithms, Performance, and Reproducibility" by SCC Team From Peking University

Yihua Cheng, Zejia Fan, Jing Mai, Yifan Wu, Pengcheng Xu, Yuxuan Yan, Zhenxin Fu, and Yun Liang

Abstract—Shi *et al.* (2018) proposed a highly parallel polynomial filtering eigensolver for the computation of planetary normal modes. As a challenge at the Student Cluster Competition in The International Conference for High Performance Computing, Networking, Storage and Analysis (SC19), we reproduce the computational efficiency of the polynomial filtering eigensolver on our Intel Xeon machine. We present the weak scalability, scaling of runtime with model size (in a fixed interval) and the strong scalability results in this report.

Index Terms—Reproducible computation, student cluster competition

1 Introduction

PLANETARY normal modes are used to provide a framework for representing global seismic waves and can support the study of internal planetary density and attenuation efficiently. The paper Computing Planetary Interior Normal Modes with a Highly Parallel Polynomial Filtering Eigensolver [1] (the Normal Modes paper) applies a highly parallel polynomial filtering eigensolver to the computation of the normal modes, achieving significant enhancement in memory and computation efficiency without loss of accuracy.

The Normal Modes paper transforms the problem of computing planetary normal modes into a generalized eigenvalue problem (GEP), then performs sparse matrix-vector multiplications (SpMVs) to solve the math problem, and finally applies a Lanczos approach to calculate the normal modes from the GEP. Three parallelism optimizations for solving this GEP are proposed:

1) Slicing spectrum into intervals and applying parallel polynomial filtered Lanczos solvers to each interval respectively;
2) domain decomposition for better memory scalability;
3) parallelization of the communication and the local computations of SpMVs.

In this report, we reproduce a set of experiments to evaluate mainly the scalability of the polynomial filtering eigensolver. The results are listed below:

1) two sets of weak scalability experiments, showing the runtime of the SpMV implementation with increasing model size;
2) scaling of runtime with model size in a fixed subinterval;
3) strong scalability, showing the runtime and parallel efficiency with the increasing number of processes.

- *Yihua Cheng and Yifan Wu are with Yuanpei College, Peking University, Beijing 100871, China. E-mail: {chengyihua, yifan.wu}@pku.edu.cn.*
- *Zejia Fan, Jing Mai, Pengcheng Xu, Zhenxin Fu, and Yun Liang are with the School of Electronics and Computer Science (EECS), Peking University, Beijing 100871, China. E-mail: {zejia, magic3007, jsteward, fuzhenxin, ericlyun}@pku.edu.cn.*
- *Yuxuan Yan is with the School of Physics, Peking University, Beijing 100871, China. E-mail: yanyx1999@pku.edu.cn.*

Manuscript received 7 Oct. 2020; accepted 30 Dec. 2020. Date of publication 5 Jan. 2021; date of current version 10 May 2021.
(Corresponding author: Yifan Wu.)
Recommended for acceptance by B. A. Plale and S. L. Harrell.
Digital Object Identifier no. 10.1109/TPDS.2020.3049050

The experiments are run on our 5-node server, with 2 CPUs and 36 cores on each node.

2 Experimental Setup

In this section, we describe the experimental setup for the reproducibility test, including hardware configuration and OS version, compilers and their versions, libraries and dependencies, and input datasets for experiments.

Hardware Configuration and OS Version. We use our server to perform all the reproduction experiments. The hardware configuration and OS version are shown in Table 1. We run the experiments on the 4 nodes with the same CPUs.

Compiler and Version. *Intel C++ and Fortran Compilers* 2019. It is a set of C, C++ and Fortran compilers, which also contains MPI compiler we use for this reproduction.

Libraries. *ParMetis* (version 4.0.3). It is a parallel library for graph partitioning and fill-reducing matrix ordering, used by Normal Modes.

pEVSL (SHA 7bbf93b1cf). It is Jia's modified version of pEVSL, a parallel eigenvalue slicing library, to compute the eigenvalues in Normal Modes.

Intel MKL (Intel Math Kernel Library 2019). It is a library used by pEVSL for matrix multiplication and other math operations.

Hardware and Software Difference. The Normal Modes paper runs its experiments on platforms equipped with E5-2860v3, KNL, and Skylake CPUs, respectively. We reproduce the results on our Intel Xeon Gold 6240, the Skylake architecture. Our memory capacity is a different 384G. We use the same software setup as the Normal Modes paper.

Datasets. The reproducibility tests are run on six Mars models M1-M6, with reference gravity and both fluid and solid regions. Details of the datasets are shown in Table 2. $A_G = \text{diag}(A_{sg}, A_f)$, A_{sg}, A_f and A_p are solid stiffness matrix with reference gravity, non-seismic modes in the fluid regions and fluid pressure matrices with/without reference gravity, respectively. Fig. 1 shows a visualization of the twentieth normal mode of M4.

Modifications. We did not modify the original code. The experiments are conducted with Normal Modes (SHA 157d306b66).

3 Description of Experimental Run

Table 2 shows the description of our test cases and the E3, C3 models in the Normal Modes paper. In the Normal Modes paper, the number of processes is scaled with the size of datasets. However, if we scale the number of processes together with the problem size, we are not guaranteed to finish all experiments in the 48-hour competition. As a result, the datasets are divided into two subsets (Set1, Set2) for weak scaling and scaling with model size tests, of which both are run from 1 node to 4 nodes, in order to finish the study on our 5-node server.

In the Normal Modes paper, strong scalability is studied with E3 which is 4 times the size of M2 from 4 nodes (192 processes) to 16 nodes. So we compute the normal modes for model M2 from 1 node to 4 nodes (1/4 of the nodes for E3).

All the tests are run on the 4 nodes with the same CPUs (2× Intel Xeon Gold 6240). To compile and run the Normal Modes application, we use the script `compile/compile.sh` and `run/run.sh`, respectively.

4 Weak Scaling of SpMV

As shown in Table 2, the size of elements doubles from M1 to M3, and from M4 to M6. To run the tests with 1, 2, 4 nodes on our 4 nodes with the same CPUs, we divide the dataset into two subsets Set1 and Set2. We double the number of nodes and their processes for each of the subsets and collect two sets of output (shown in Table 5).

TABLE 1
Hardware Configurations of Our Machine

5×CPU node	
Part	Model
CPU	2×Intel Xeon Gold 6240 (2×Intel Xeon Gold 6248)[a]
Memory	12×32G, DDR4
NIC	InfiniBand EDR switch GigEthernet switch
OS	CentOS 7.7 x86_64

Our machine consists of 5 nodes. The configurations of each node are shown in the table.

TABLE 2
Description of Different Models

Exp	nn/np	# of elm.	size of A_G	size of A_p
M1	1/36	591,301	281,784	7,996
M2	2/72	1,294,290	610,335	17,781
M3	4/144	2,229,825	1,208,013	42,865
M4	1/36	3,706,428	1,786,566	79,871
M5	2/72	7,532,466	3,569,283	158,922
M6	4/144	15,283,899	7,182,987	315,876
E3	-	8,000,777	4,466,349	311,655
C3 (solid)	-	8,079,387	size of A: 3,894,783	

M1-6 are the Mars models we use throughout this reproducibility task, while E3 and C3 are the earth model and the solid model, respectively, used in the Normal Modes paper.

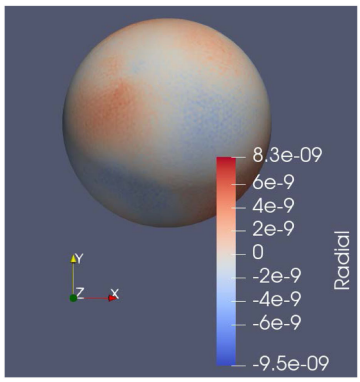


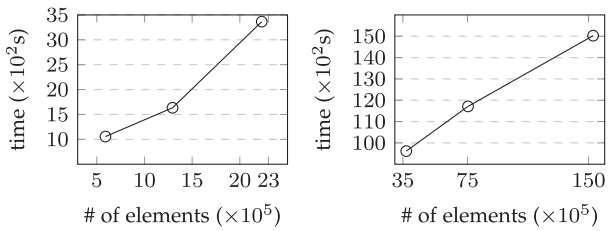
Fig. 1. A visualization for the twentieth normal mode of M4.

The model name, node/core count, number of elements and size of A_g are shown in Table 2. In each run of the algorithm, the SpMV is performed multiple times. Table 3 shows the average times for performing a SpMV with matrices A and M in one run.

Difference. From Fig. 2 it can be seen that for Set1, the SpMV implementation does not show good weak scalability, while for Set2, the parallel efficiency is good above 0.89. We notice that for all datasets in the Normal Modes paper, the size of A_G/ np is around 12000. For tests with Set1 and Set2, the size of A_G/ np is around 7800 and 49600, respectively. A possible explanation for the bad scalability on Set1 is that the size of the matrix in SpMV is too small for the number of processes so that the communication time cannot overlap well with local computation as shown in alg. 1 in the Normal Modes paper.

TABLE 3
Test Cases for Different Mars Models

Set	Exp	T-Av (s)	eff.-Av	T-Mv (s)	eff.-Mv
Set1	M1	0.001946	1.0	0.000223	1.0
	M2	0.003460	.56	0.000400	.58
	M3	0.006534	.30	0.000572	.39
Set2	M4	0.013974	1.0	0.002237	1.0
	M5	0.014817	.94	0.002324	.96
	M6	0.016818	.83	0.002508	.89



(a) Total runtime of Set1. (b) Total runtime of Set2.

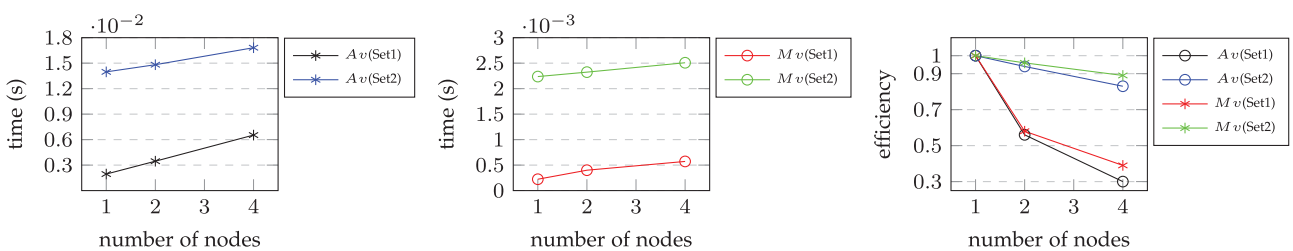
Fig. 3. The total runtime for experiments on Set1 and Set2.

5 Scaling With Model Size

We use the results from the same experiments in Section 4. In Table 5 we show the degrees of the polynomial filter and the numbers of iterations.

Difference. The number of iterations to convergence are the same for all the models. The results in our paper show that for some small datasets, the number of iterations is different, and others are the same. It can be explained by the difference in interval length in the experiments. The Normal Modes paper uses an interval $(3.9e-7, 8.9e-5)$, while here we use the same interval $(9.9e-06, 3.9e-05)$, smaller than the target interval. The degree of the polynomial filter is increasing in Set2, which is similar to that shown in the paper. However, for Set1, it is abnormally low for M2.

In Fig. 3 we show the total runtime for Set1 and Set2. For Set2, the curve is concavely going up, similar to that in the Normal Modes paper. However, for Set1, the curve seems to be convex. We show a possible explanation of this difference in Section 4. It can probably also be explained by the abnormally low degree of polynomial filter for M2.



(a) Parallel SpMV time for Matrix A. (b) Parallel SpMV time for Matrix M. (c) Parallel efficiency.

Fig. 2. Strong scalability plot for the M2 model.

TABLE 4
Strong Scalability Tests for Problem M2

nn	T-Av (s)	T-Mv (s)	T-$M^{-1}v$ (s)	total (s)	eff.
1	0.003802	0.000645	0.017186	2585.00	1.0
2	0.003460	0.000400	0.010378	1634.10	.79
3	0.002700	0.000241	0.006234	1077.31	.80
4	0.003223	0.000229	0.005819	1046.32	.62

On each node, 36 processes are run (i.e. $np = 36\times$ nn).

TABLE 5
Test Cases for Different Mars Models

Exp	nn/np	# of elm.	deg	#it
M1	1/36	591,301	534	172
M2	2/72	1,294,290	517	172
M3	4/144	2,229,825	794	172
M4	1/36	3,706,428	569	172
M5	2/72	7,532,466	702	172
M6	4/144	15,283,899	892	172

6 Strong Scaling

As described in Section 3, since the M2 model is 1/4 size of E3 in the Normal Modes paper, we use 1/4 number of processes of that in the paper to perform the strong scalability study.

Our results are shown in Table 4, including the average time of calculating an SpMV on A and M in one run of the Normal Modes algorithm, the average time of solving M with Chebyshev iterations in one run, and the total time for computing the wanted eigenvalues. We plot the total runtime of models C3 and E3 in the Normal Modes paper, and of our M2 in Fig. 4. In Fig. 5, we show the parallel efficiency on these data sets in two plots.

Difference. On our model M2, the parallel efficiency is generally good. However, the parallel efficiency drops from 1 to around 0.80 when we increase the number of nodes from 1 to 2 and 3, while in the paper, the curve stays high around 1.0 when the number of nodes is doubled. A possible reason is that from 1 node to 2 nodes, intercommunication is required between nodes. Our dataset is much smaller than the C3 and E3 dataset, so the communication cost is larger.

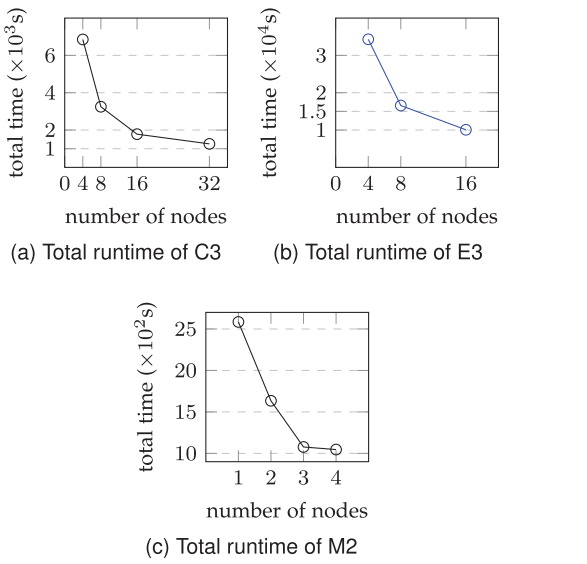
When we increase the number of nodes from 2 to 3, the parallel efficiency goes up. One possible explanation of the abnormal strong scaling efficiency when the node count is low has its roots in the nature of hierarchical caches in modern computing systems. The execution time of any parallel task can be modeled as

$$T(m,n) = \sigma(m) + \frac{\phi(m)}{n} + \kappa(m,n),$$

where m is the problem size factor and n is the processor count. T is the overall execution time, while σ, ϕ, and κ are the respective serial and parallel execution time and communication cost. In the case of strong scaling, the problem size m is fixed, after which we can model the execution time as

$$T(n) = \sigma + \frac{\phi}{n} + \kappa(n).$$

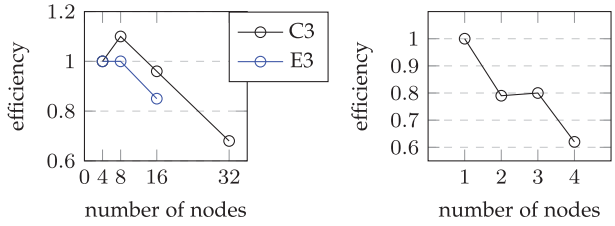
Modern systems use hierarchical caches to improve performance. Due to the fact that the serial and parallel portions of the task run on the same processors, they share the same number of cache lines, leading to σ being inversely related to n, and thus the efficiency being greater than 1 when $n > n_0$. However, the serial speedup effect is bounded by the total cache lines available on a single processor, while the communication cost κ usually grows with n without an upper bound. This results in the efficiency getting below 1 when the processor number gets bigger.



(a) Total runtime of C3 (b) Total runtime of E3

(c) Total runtime of M2

Fig. 4. We plot the total runtime of models C3, E3 in the Normal Modes paper, and our model M2 for comparison.



(a) Parallel efficiency of C3 and E3 (b) Parallel efficiency M2

Fig. 5. We plot the parallel efficiency of models C3, E3 in the Normal Modes paper in plot (a), and our model M2 in plot (b).

7 Conclusion

In our paper, we reproduce the scalability performance of the Normal Modes paper. We provide the details of our hardware and software configuration, compiler and versions, dependencies and datasets used for the experiments. We describe how our experiments are designed and how they are performed. Our comparison on the weak scaling and scaling with model size study shows that the methods proposed in the Normal Modes paper work well with large datasets and a suitable number of processes. On relatively small datasets and a large number of processes, however, the scaling performance is not as good as that in the paper. The strong scalability result shows the method achieves good parallel efficiency. In conclusion, we are able to reproduce the claims of scalability in the Normal Modes paper.

Acknowledgments

The authors would like to thank the hardware and technical support from their sponsors Inspur, Nvidia, and SitonHoly. Yihua Cheng, Zejia Fan, Jing Mai, Yifan Wu, Pengcheng Xu, and Yuxuan Yan contributed equally to this work.

Reference

[1] J. Shi, R. Li, Y. Xi, Y. Saad, and V. Maarten, "Computing planetary interior normal modes with a highly parallel polynomial filtering eigensolver," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2018, pp. 894–906.