Dear editors and reviewers

Thanks a lot for your scores of suggestions about this manuscript. Per your comments (in black font), we have revised our manuscript accordingly (in red font) and made point-to-point responses (in blue font) to all the comments and concerns. Below are our detailed responses.

Referee #2,

This manuscript presents a refactoring of some routines of WRFPLUS in order to decrease the execution time maintaining the total memory required.

Reply: Sorry, no complete code is uploaded during the first manuscript. Here, we move the complete code to a new site (https://github.com/juanjliu/WRFDA-OPTIMAZATION).

More detail information about the usage can be found from "README.md" on https://github.com/juanjliu/WRFDA-OPTIMAZATION.git.

The complete code is helpful to understand "Refactoring is based on several techniques and full using of the conception "adjoint locality"".

1 General comments

Authors state that they have used several techniques (push/pop-free method, IO analysis, use of adjoint locality) but my first impression is that the paper consists only on a code refactoring affecting some routines. This refactoring was basically done replacing superfluous subroutine calls with push/pop operations and modifying the way that push/pop operations are done. Even if they obtain a significant reduction in the execution time, I do not see in this work a real scientific contribution, but rather an engineering practice.

Reply: We just say we could do nothing about these code optimizations without the direction of the conception "adjoint locality". And how valuable is the push/pop-free method directly decided from practice instead of anything else.

In the example shown in Figure 2, there is not any superfluous subroutine calls without our consideration with adjoint locality. Generated the adjoint code of a subroutine/function with most AD tools including TAPENADE, a full sweep/duplication is always required before local adjoint accumulating, which is clearly shown in Figure 1. And this is the fundament fact of some checkpointing methods. However, such weep is not always indispensable to adjoints in a sense of adjoint locality.

We present all modified code, you will find that we just purely using the push/ pop-free method within the core adjoint subroutine solve_em_ad. Within the adjoint code optimization of WRF, we scarcely used such replacement which is extracted from another application the GPS/MET rayshooting model as shown in Figure 2. However, we in most cases decrease most

execution time in most adjoint routines just by changing its computational sequence with the help of the understanding of adjoint locality.

Actually, the word "SCIENCE" is so noble instead of anything with only a few modifications or several skills. We never thought there was scientific contribution in this article, but we do believe the techniques and the conception discussed are really valuable in automatic differentiation fields. And we were told that new techniques and new methods for code improvement were welcomed by GMD. If you compare our results with those from other AD fellows, you will find that it is so difficulty to require such less adjoint execution cost without any/extra memory increment at the same time.

The paper is difficult to read and understand. A general review in order to improve the quality of the writing is necessary.

Reply: English is really a big problem to us although we have tried to make it clear so many times for readers. We wish there is a fellow who is interested in this article from English-speaking country to join in the author list.

2 Specific comments

• About the Input/output analysis. Usually, I/O refers to disk or network accesses, but no memory accesses. In the paper it is difficult to see if a real I/O analysis has been carried out.

Reply: I agree with you. IO here refers to "Input and Output" of a program object such as a subroutine, which has clearly defined in the abstract part.

• P2, L6-7: "To some AD applications such as 4DVAR, much more memory consumption must result in poor scalability both in computing grids and in different computer environments." Why is this true? What do you mean by "computing grids"?

Reply: "computing grids" means the resolution of model. For example, the model state x_i is a solution of model equations:

$$\forall i, \qquad x_i = M_{0\to i}(x_0)$$

where $M_{0\to i}$ is a predefined model forecast operator from the initial time to i. For the second case in the manuscript, we use 270×180 horizontal grids and keep the 41 levels, focus on 5 model variables and the integration time step of 180 seconds, then the dimension of the basic state (x_i) at one step is 270×180×41×5. If the forecast time is 12 hours, there will be 240 steps (x_i) to save. For example, WRFDA successively pushes the required data $x_1, x_2, \cdots, x_i$ into a huge stack L, then it needs huge costs in memory.

So, as the number of computing grids increases, the extra memory for running the adjoint model could be out of the available memory.

• P2, L31-32: ". . . by a slight of hand several years ago", What does this expression mean?

Reply: It should be “with a slight of revision”. Thanks.

• P2-3, L34-1: “Actually, the computational cost for these push/pop operations will turn out to be comparatively expensive when the local adjoint cost is reduced at a lower degree.” I suppose that you are talking about the percentage of the push/pop operations over the total cost, aren’t you? Please clarify it.

Reply: The computational cost of these push/pop operations are negligible when there is no accompanying adjoint optimization, but if the adjoint model is optimized and the ratio between the adjoin model and Nonlinear model reduces from 5:1 or 6:1 to 3:1 or even lower, the percentage of the push/pop operations can’t be ignored.

For example, you cut 1 from 10 and you can get 10% improvement. However, if you cut 1 from 3 or less and you can get more than 33% improvement)

• P3, L8-10: “. . . the most difficult optimization may be associated with huge memory consumption stemming from the fact that the computational cost of the core adjoint procedure is almost evenly distributed across its calling subroutines. . . ” Why is the memory consumption related with the cost balance of the subroutines? Please, clarify.

Reply: “which resulting in scores to a hundred of push/pop operations inserted between these calling subroutines and their adjoints are required within each call of the core adjoint procedure for less execution time.” is added in the revised manuscript. However, an opposite case such as presented in Figure 2 requires less memory, in which the computational cost of the considered subroutine is concentrated on one or only a few calling subroutines.

• P3, L16: “global/local IO", What do you mean by global and local IO in this context?

Reply: A variable may have different IO attribute during different program objects as well as different local segment of program lines.

• P3, L26. You should specify that X and Y _ are in this context.

Reply: Thanks a lot. “where δX and δY^* are respectively the input of both differential models.” is added in the revised manuscript.

• P4, L18 and Fig. 1: Figure 1 does not help to understand the concept of “adjoint locality". A better explanation of this figure is necessary.

Reply: Yes. Figure 1 is only a typical push/pop strategy for most adjoint implementations from practical applications through which we present our basic ideas about “adjoint locality”, we add a general example of subroutine/function splitting for this suggestion.

• P5, 2nd paragraph: It is said that required data are pushed into stack “. . . during each sweep of RP when running its adjoint subroutine Adj_RP. . . ". In the same paragraph: “However, the sweep of RP within Adj_RP will be completely removed for better adjoint performance. . . ” If the sweep of RP is removed, how are data pushed into the stack? Please, clarify.

Reply: It is clear to show this from the last sentence “In Figure 3, we therefore only keep a very small part of the required data directly from the first run of the root subroutine RayFind with the extra memory….”

• Figure 3: How can the second call to LL2JK be removed if P is not popped out from the stack?

Reply: That is because Int3SL calling has been removed in Figure 3 such that the value of P was never changed not as in Figure 2.

• P5, L16: Please, indicate what NV is.

Reply: Yes, Thanks.

• P6, L19-21: “First, we use several or more stacks instead of only one, each of which is still a 1D data structure. It has been shown that this type of data structure has the advantages of smaller access cost and flexible expressions, either in the communications between procedures or in local program calculations.” Can you please add any reference to justify this assertion?

Reply: Thank you very much for your suggestions. But in fact, we just speak this from our practice with WRFPLUS. Based on careful consideration, we add “in practice” into this statement Instead of adding a reference.

• P6, L23. “Although the cost of allocating/deallocating many dynamical stacks cannot be neglected compared with the costs of these adjoint procedures. . . ” Again, this assertion should be justified. Which memory allocator are you using?

Reply: “…allocating/deallocating many dynamical stacks in Fortran 90/95 programs…” is added in the revised manuscript. Thanks.

• Section 3. In the paper, push/pop operations are replaced by copying the data into a vector (keepx) and then recovering them from that vector. I cannot see how different this data movement is from using a stack. In the paper it is said (P7, L14-16) “Note that the pushing operation PUSHREAL8ARRAY is replaced by a direct evaluation statement from x to keepx with approximately the same cost both in run time and in memory.” So the only improvement is due to the replacement of the pop operation. Might not be simpler to optimize the use of this operation?

Reply: Several similar questions have been asked. Thanks. Therefore we add something such as “Actually, the computational cost for these push/pop operations will turn to be comparatively expensive in some cases such as when the push and pop operations are located in different innermost loops or if the local adjoint cost is reduced at a lower degree.” to the introduce part. 1) Above says the cost of push/pop operations cannot be overlooked in some cases. 2) Only the pop operation is saved in corresponding to this case when push and pop operations are located in different subroutines or functions, the past answer is “Statistically both the push and the pop operations can be improved in this way in most adjoint codes. Less adjoint implantations use a pair of push/pop operations in different subroutines/functions.” And we have presented “Another case is **infrequently** used in some adjoint implementations that the push operation and the corresponding pop operation are located in two different subroutines or functions in” in the last part of Section 3. 3). It is really not easy to use this method everywhere

such that we actually have revised only less than 5% of the total push/pop operations in those adjoint subroutines/functions. However, that is enough to have an obvious improvement of the adjoint model.

• P7, L15: ". . . a direct evaluation statement from x to keepx. . . " Please, clarify

what do you mean by "a direct evaluation", isn't it a simple copy?

Reply: Yes, it is. That means only the pop operations can be saved in such cases. "…a simple duplication from x to keep_x…."

• P7, L28. "sulfurous calculations." I suppose you mean "superfluous".

Reply: Yes, Thanks.

• Regarding the test results, more details about how the experiments were carried out should be included, e.g., compiler used, optimization level, MPI version, characteristics of the cluster (cores per processor, memory per core, etc).

Reply: All experiments were conducted on LINUX and a Distributed Memory (DM) server cluster with 250 nodes connected with InfiniBand, each of which has 20 Intel Xeon E5-2670/2.5G processors and shares 62 GB memory. Compiler used is Intel-ifort version 13.1.1, optimization level is O3, and mpiifort for the Intel(R) MPI Library 4.1 for Linux. All information have been added in the revised manuscript.

We will upload the configure as a supplementary material.

• Table 1: You use until 256 processors, in a system with 5000 processors (250*20). It will be interesting to see values for a larger numbers of PEs and a larger resolution.

Reply: WRFPLUS will lose a lot of parallel performance for practical applications when the number of processors is more than 512. And we scarcely have chance for using more than 512 processors at present.

• Table 1: How many experiment have been done for each value of PE? Are the results in the table an average? What is the standard deviation?

Reply: This is a big problem. Each data of the time cost for the core adjoint procedure solve_em_ad in Table 1 and Table 2 is picked up from scores to hundreds of data, almost an average of those data. For example, the core adjoint procedure solve_em_ad needs to execute for scores of times within each calculation of gradient, but we just select one that is probably most near to the average. So we revise it into "Table 1 shows the **average** wall-clock time for one-time-step integration….."

• P8, L11-12: "At the same time, both aspects of improvements are slightly increased with the increase of the number of processors. . . " What are those "both aspects"?

Reply: Thanks. So "…both aspects of improvements for the adjoint model and the 4DVar system…"

• P8, L18: ". . . uniformly different by no more than two last significant digits in double

precision." If you do not mean the last two bits in the double precision binary representation, this statement is meaningless. You should use the absolute difference between values.

Reply: The correctness of the adjoint model can be checked by the following algebraic expression:

$$(M_r z)^T (M_r z) = z^T ({M_r}^T (M_r z)) \quad \text{(R1)}$$

Where M_r is the tangent linear model of nonlinear model, ${M_r}^T$ is the adjoint model. If the code is correct, equation R1 should be verified for all input values of z. Compare the values of left and right to see whether they are equal to the machine accuracy. 13 digit accuracy is the best result to be expected. In some rare cases, less than 13 digit accuracy can still represent a none-error adjoint code. A lot of experiments show that a minimum of 9 digit accuracy are required for a single-precision machine (see NCAR Technical Note, 1997, NCAR/TN-435-STR). For example, our checking result is the following:

VAL_L: 0.23958449830151E+10

VAL_R: 0.23958449830152E+10

I agree with that it is not clear here. So, we revised it into "The reliability of the optimized version has first been checked in terms of the correctness test **(Zhang et al. 2013)** in many applications, each of which is uniformly different by no more than two last significant digits in double precision **with the underlying version**."

Reference

Zhang, X., Huang, X. Y. and Pan, N.: Development of the Upgraded Tangent Linear and Adjoint of the Weather Research and Forecasting (WRF) Model. J. Atmos. Ocean. Tech., 30(6), 1180-1188, doi: 10.1175/jtech-d-12-00213.1, 2013

• Test result section: It will be very interesting add some graph to show whether any change in scalability (both strong and weak) is caused by the proposed improvements.

Reply: Thanks. But in some cases, it is very difficult to clarify a change is whether using the push/pop-free method or other techniques in terms of adjoint locality. So for this comment, we believe that there will be a lot of extra work to do.