Geosci. Model Dev. Discuss.,
https://doi.org/10.5194/gmd-2020-75-SC1, 2020

Geoscientific Model Development Discussions

Interactive comment on "Developing a common, flexible and efficient framework for weakly coupled ensemble data assimilation based on C-Coupler2.0" *by* Chao Sun et al.

Lars Nerger

lars.nerger@awi.de

Received and published: 29 May 2020

Dear authors,

thank you for this interesting article on the assimilation framework based on C-Coupler.

I am Lars Nerger from the Alfred Wegener Institute in Germany and leading the development of PDAF, the Parallel Data Assimilation Framework. I'm happy to see that our work on PDAF motivated your work so much.

Having read the article, I have the impression that there are some misunderstandings about the functionality and requirements of PDAF. Apparently not all aspects have been sufficiently clarified in the response of my coauthors and me to Dr. Liu's comment on our article "Efficient ensemble data assimilation for coupled models with the Parallel Data Assimilation Framework: Example of AWI-CM" (https://doi.org/10.5194/gmd-2019-167). Thus, I like to clarify these points so that they can be correctly described in your article:

In lines 84-86 you write *"PDAF addressed the problem of generating local communicators by imposing a precondition of process layout such that each ensemble member uses the same number of processes with successive IDs in the MPI_COMM_WORLD."*
PDAF does actually not require this precondition of process layout. It is not required that the processes have successive IDs in MPI_COMM_WORLD. With PDAF we provide the routine init_parallel_pdaf to perform the initialization of the MPI communicators. This is a template file, which can be adapted to the users' needs. The default configuration uses successive IDs, because it's simple to implement and consistent with most process layouts used in the models. However, this can be changed by the user, and an arbitrary ordering of the IDs would be possible (for typical cases like ocean or atmospheric models this would hardly be useful, but we cannot exclude that there are models which benefit from such distribution of the processes). It is also not strictly required that each ensemble member uses the same number of processes. This is the default setup, which should be a reasonable choice as running different ensemble members with different numbers of processes likely results in different execution times and hence some processes would need to wait for others when it comes to the analysis step of the ensemble data assimilation.

In lines 86-88 you write *"For example, if there are 10 members in an ensemble run and each ensemble member uses 50 processes, there should be 500 processes (#0-#499) in the MPI_COMM_WORLD, and processes #0-#49 will be used for the first ensemble member"*
It follows from the previous clarification that the statement on processes #0-#49 only

holds for the default setup, but not in general. Further, it is not required that there are 500 processes, thus enough processes to integrate all ensemble states in parallel. PDAF provides the flexible parallelization variant, in which, e.g., the 10 members of the example could be run using 250 processes, or even only 50 processes. This parallelization variant is described in the PDAF documentation on the PDAF web site http://pdaf.awi.de and in Nerger et al. (2005).

In lines 88-90 you write ***"Such a solution can facilitate the corresponding code implementation, but requires the user to know and guarantee the process number precondition when submitting the MPI job of an ensemble run."***
As described before, with PDAF one does not need the full amount of processes. Of course one does need to know the number of processes and needs to ensure this number when submitting an MPI job. This is the case for any MPI execution since one needs to specify the number of processes in the call to mpirun (or mpiexec/aprun/srun) and for supercomputers with batch systems one needs to know this number when specifying the resource requirements for a compute job.

In lines 98-100 you write ***"... the user is still required to perform many tasks; e.g., developing the call back functions for gathering the same model field from all ensemble members into a vector of the DA method and distributing the data of a field in a vector to all ensemble members"***
It is correct that such operations are required. However, implementing them is a pretty trivial task. Actually, this operation is always model-specific and an analogous operation even happens when using a coupler like C-Coupler. However, in this case it's in the coupler API instead of the assimilation API.

In lines 101-103 you write ***"The user is recommended to make the DA implementation using the same parallel decomposition (grid domain decomposition for parallelization) as the model, and needs to develop new code when the model and DA implementation use different parallel decompositions."***
In deed we recommend to use the same domain decomposition. To our experience this

makes it easier for the user to implement the call-back routines. Nonetheless, PDAF allows a user to use more processes for the analysis step, but then the user has to ensure that the model fields are properly distributed.

In lines 103-104 you write ***"... efforts should be made to enable the software compilation system of the model to compile the code of the DA methods."***
This is actually not correct. One usually compiles the PDAF core library, which contains the code of the DA methods, separately and later links this library with the model code. Here, it's not relevant if the PDAF library is compiled as a static or dynamically linked library (This seems to be rather a matter of personal taste, even though for some supercomputers like the Cray XC series the use of dynamically linked libraries is not recommended). For the user-code (the call-back routines), we recommend to compile the code with the model. However, this is not strictly required and only could also compile this code separately as a library. Anyway to our experience it's easier to compile the code with the model, because it helps to ensure a consistent compilation.

In lines 108-110 you write ***"Although PDAF enables a DA algorithm to run in parallel, it only makes the processor cores of the first ensemble member available to the DA algorithm and forces the processor cores used by other ensemble members to idle when running the DA algorithm."***
As described above in relation to lines 86-88 and lines 101-103, it is not correct that PDAF "only makes the processor cores of the first ensemble member available to the DA algorithm". This is just the default configuration, which can be adapted. However, to our experience the analysis step, i.e. the execution of the DA algorithm, is usually fast compared to the time to compute the ensemble forecast. Thus by letting more processes compute the DA algorithm, the overall speedup will be limited. In contrast, using more processes for the analysis step requires remapping of the domain decompositions. This requires more MPI communication calls, which will take more time than just collecting ensemble information on the first ensemble task. Thus, while one gains speed in the analysis one looses time in the remapping. What is faster will

be case-dependent.

I hope that the explanations above help to correct the statements about PDAF in your manuscript. Overall, I have the impression that one has to be very careful when claiming limitations because the risk to be not fully accurate is high. From reading your article I have the impression that it rather presents a different strategy to couple the DA algorithm with the model, than one that overcomes limitations. PDAF provides a lightweight coupler, particularly aimed at data assimilation, which can be easily connected to a model, even if the model already uses a model-coupler (like OASIS, which is implemented in the AWI-CM model used by our study mentioned above). In the framework described in your article a complex coupler is used, which can, e.g., perform mapping between different domain decompositions. However, using this coupler will e.g. require to implement subroutine calls that describe the model grid so that this information can be handled by the coupler and the by DA algorithm (When I read this correctly in the source code you provided with the article, this is done in module_wrf_DA.F which contains about 56 calls to procedures of the coupler). Thus, it seems to be more work to add the coupler functionality of C-Coupler to a model than just writing model fields into a vector as is done in PDAF.

Overall, as the limitations of PDAF, which are claimed in the article, do actually not exist (all three bullets of the list at the beginning of Section 3 are also fulfilled by PDAF), I would find it only suitable to describe the framework based on C-Coupler as a strategy that is 'different' from that of PDAF, rather than a strategy that solves apparent limitations.

Kind regards,
Lars Nerger