General Comments:

This paper aims to introduce the CCPP framework that is for flexibly using atmospheric physical parameterization schemes in various model simulations. I think it deserves attentions from model developers, model users and framework developers, as it presents new thoughts for integrating and using parameterization schemes, and the CCPP framework has already been used in real model development. However, I think the current version of this paper should be significantly improved after revisions. The current context seems hard to understand, as it focuses on the design and implementation of the CCPP framework, without sufficient motivations and analysis that are important to make this paper more attractive and understandable.

Specific comments:

1. About motivations of developing the CCPP framework.

It will be welcome to give examples for motivating this framework, e.g., the current status of physical packages, real scientific requirements that the current physical packages cannot serve but the CCPP framework can.

2. About motivations of the design and implementation of the CCPP framework.

It seems that the main target of the CCPP framework is to enable users to flexibly group physical schemes into a package via an XML configuration file (Listing 5). To achieve such a target, authors prefer a solution that automatically generates Fortran code of callers corresponding to the configuration file. A challenge here is how to achieve automatic caller code generation, as the generated caller code should match the argument list of the corresponding callee that is a physical scheme. To overcome this challenge, authors propose to describe each argument of a physical scheme via the corresponding metadata (listing 2), including the variable name, dimension order, data type, in/out attribute of each argument. Moreover, authors seem to still keep the rule that a caller and a callee share the same data structures and the same variable name space of all arguments, and parse argument values between each other via shared memory space. Under such rule, a kind of metadata is designed for generating the Fortran code of declaring all model variables that are shared by all physical schemes.

Under the above understanding (I am sorry if I misunderstand), I think this paper should state why the current technical solution of the CCPP framework is almost the best option. I am a little afraid that the current solution that highly depends on automatic code generation and the development metadata may introduce new challenges to users, e.g., work for studying the metadata rules and developing metadata files, more phases in compiling the codes, and efforts for guaranteeing the consistency between the metadata files and codes.

There may be other possible solutions. For example, there can be native Fortran code files for declaring all variables and data types used in the physical package, a native Fortran code file that enumerates all callers each of which corresponds to physical scheme, and a native Fortran

code file for controlling the usage flow of a set of physical schemes under the XML configuration file. Even when a Fortran code file is not flexible enough for achieving such a control flow, C/C++ can work cooperatively with simple Fortran callees with a few arguments (a complex Fortran callee can be enclosed in a simple callee that parses arguments to the complex callee via global data). Such a simple design may not achieve all goals of the CCPP framework. However, I believe that comparisons with simple designs can make this paper more understandable.

3. About "Neither Fortran common blocks nor the local definition or importing of physical constants or functions from other physical parameterizations are allowed. Physical constants must be passed to the physics schemes via the argument list. Each CCPP scheme may define its own dependencies (i. e., Fortran, C, C++, . . . modules) that are required for the scheme, as long as these do not depend on the presence of other schemes and vice versa." and "To ensure a consistent set of physical constants for use by all physical parameterizations, these constants must be defined by the host model and passed to the physics via the argument list, in other words, they are treated like normal variables. "

   I cannot fully understand the necessity of the above restriction rules. I tend to guess that a scheme should not share any global data (including constants) with other schemes and the model, and all arguments of a scheme should be passed via the argument list. However, the model and different schemes essentially share the same data structures and the same variable name space of all arguments. Moreover, it seems difficult for the CCPP framework to detect the case of sharing global constants as well as global variables among different schemes.

4. About Figure 2 and Listing 1.

   Can authors give an example about the corresponding motivation? When the blocked data structure is initialized from a round-robin parallel decomposition and a block contains multiple columns, a contiguous array generated by the code in Listing 1 does not match the horizontal grid. As each scheme share the parallel decomposition of the model, is there any restriction about the parallel decomposition?

5. About dynamically linked library (DLL).

   I note that an old version of CCPP used DLLs to enclose schemes. Does the new version still use DLL technique? If not, it will be welcome to introduce the corresponding reason in revisions.

6. Can users use CCPP in a recursive manner, e.g., a scheme works as the host model of other schemes? If cannot, does CCPP has the functionality to forbidden the recursive usage?

7. Can CCPP be used in an incremental manner, e.g., a part of schemes are used with CCPP while a part of schemes are called in a traditional manner? If can, is there any restriction?

8. About sharing the same schemes among different models.

It will be welcome to discuss about this question. Given that a set of schemes have been adapted to CCPP and used in model A, what efforts should be made for using these schemes with CCPP in model B that has the variable names (as well as the dimension order) different from the model A?