Dear editor and reviewer,

First of all, we would like to express our sincere appreciation to your valuable feedbacks. Your comments are highly insightful and enable us to substantially improve the quality of our manuscript and OpenArray. Below are our point-by-point responses to all the comments and our plans to revise the manuscript.

Responses to the comments of referee #1

1 General Comments

While the discussion paper is well structured and clearly written I am missing some parts which I outline below. In my opinion the publication as a whole does not need to be restructured or rewritten but I suggest to extent/rewrite/restructure the introduction by describing the state of the art in somewhat more detail. I wished the authors had mentioned their solution for IO and provided a discussion section to share their experiences and opinion about the pros and cons of their approach.

[Response]:

We appreciate for your helpful suggestions. First, we will restructure the introduction and describe the state of art in more details, especially by adding comparisons to similar work. Second, we will provide more details of the I/O frameworks, including implementation method and future improvement plan. Lastly, we will add a discussion section to present our experiences and opinion about the pros and cons of our approach.

Unfortunately, I did not succeed to install OpenArray on an OSX system (macOS 10.14.3, Armadillo 9.2, Boost 1.66, LLVM 7.0, gcc/g++/gfortran 8.3, openmpi 3.0).

[Response]:

Sorry to hear that. First, we would like to repeat the following requirements.

Before compiling OpenArray, the following dependent libraries are required:

- Fortran 90 or Fortran 95 compiler.
- gcc/g++ compiler version 6.1.0 or higher.
- Intel icc/icpc compiler version 2017 or higher.
- GNU make version 3.81 or higher.
- Parallel NetCDF library version 1.7.0 or higher.
- Message Passing Interface (MPI) library.
- Armadillo, a C++ library for linear algebra & scientific computing, version 8.200.2 or higher.
- Boost C++ Libraries, version 1.65.1 or higher.
- LLVM compiler version 6.0.0 or higher.

After the required libraries are installed successfully, checkout to branch dev and type “./test.sh" in the home directory of OpenArray, the source code will be generated in the

build folder. Then, change the directory into build, type “make -f makefile.intel oalib_obj”, then libopenarray.a and openarray.mod will be generated if there is no other problem.

According to the information you provided, a Parallel NetCDF library and a Message Passing Interface (MPI) library are needed. If you have any question, please contact us at any time.

1.1 Introduction

While the general motivation to start the OpenArray approach is made clear in this paper I am missing a more complete discussion of the state of the art. A few approaches (ATMOL, ICON DSL, STELLA) are listed but the text does not provide useful hints in how far OpenArray really goes beyond existing approaches. I am missing ATLAS (DOI: 10.1016/j.jcp.2017.07.006) . I am not an expert in this field, but to me ATLAS seems to cover several design aspects, in particular operators, support for parallelism, and support for different grid types, and seems to be in these aspects similar to OpenArray. The ESCAPE project and its follow-up ESCAPE2 worked or will work in this direction. The authors cite Lawrence et al., 2018 but only as a reference for a trend towards the usage of “heterogeneous and advanced computing platforms”, even though Lawrence et al. also discuss software approaches to address these challenges, including concepts like those used by OpenArray.

[Response]:

Thanks for your nice suggestions. We will rewrite the introduction part, introducing more details about the related work, including ATMOL, ICON DSL, STELLA and ATLAS. In addition, we will compare OpenArray with them in detail to demonstrate the advantages and disadvantages of OpenArray in the revised manuscript.

1.2 IO

As a model without IO is pretty useless it would have been nice to have read a few lines about how the (parallel) IO is approached. It is included in OpenArray, so why not spending one paragraph on such an important issues as well, perhaps with some graph showing the IO performance.

[Response]:

Thanks for your valuable suggestions. We will add an introduction to the existing IO frameworks, implementation methods and performance. Actually, at present we encapsulated the PnetCDF library to perform IO functions. However, in previous work, we developed CFIO, a fast I/O library for climate models published in Geoscientific Model Development as well (X. Huang, doi:10.5194/gmd-7-93-2014). In the future, we plan to merge CFIO into OpenArray to obtain better IO performance.

1.3 Discussion

You are convincing in the sense that your approach is valid and takes major burden from the oceanographer who “only” wants to run and modify an ocean model. On the other hand the complexity has not magically disappeared but it is moved from GOMO (in this example) to OpenArray. When porting the whole software onto a new system the major effort now goes into OpenArray - which is fine, but it has to be done. How complex is this? How flexible is the OpenArray approach in this respect.

[Response]:

Thanks for your helpful comments. As you pointed out, we moved the complexity from GOMO to OpenArray. Thus, the major burden of code porting is taken away for the oceanographers. OpenArray is designed to support multiple hardware platforms through separating hardware-dependent functions such as low-level numerical computations from the main framework. In section 3.4, we will provide more details to describe the complexity of porting OpenArray to Sunway platform, including methods, workload, difficulties, etc. In the future, OpenArray will support more categories of hardware platforms.

If the unlucky oceanographer comes up with the idea to try out yet another (perhaps higher-order) advection or any other scheme which is not yet supported by OpenArray, how difficult is it to extend OpenArray? Does this require expert knowledge and support from OpenArray developers?

[Response]:

Thanks for your comments. In the current version of OpenArray, we implemented the functions of 12 basic operators and the rules for generating fusion-kernel. If users want to extend OpenArray, support from OpenArray developers is required at present. In our future work, OpenArray will be extended to support the customized operators. The definition of operators will be written in the configuration file, and the pre-compiled template will be used to generate the operator function. After that, adding the customized operators no longer require expert knowledge and support from OpenArray developers.

How seamless is - in your opinion - the integration of other stencils into the OpenArray library?

[Response]:

We use a template mechanism to implement the stencil operators, in which the rules of stencil operations are defined in a separate file, so it is easy to generate any kind of stencil kernels. The difficult part is the communication pattern among processes, for example C[i] = A[i] + B[i-2], an efficient mechanism to manage the data transfer for customize stencils are yet to be developed. The integration of other stencils into the current version of OpenArray library still requires the support of OpenArray developers. The customized operators are under development. In the future, the next version of OpenArray will support customized operators, other stencils will be conveniently

integrated into OpenArray even without the support of OpenArray developers.

I am not insisting on answering these questions line by line but rather take these as suggestions for what could be addressed in a thorough discussion. You as authors may wish to stress different - and in your opinion more important - points.

2 Specific Comments

Line 39 and elsewhere: consider to replace “climate model” by “Earth system model”, as the latter is now mainly used when talking about multi-component models in the context of Earth system and climate modelling efforts.

[Response]:

Thanks for the suggestion. In the revised manuscript, we have replaced “climate model” by “Earth system model” in Line 18, Line 31, Line 39, Line 52, Line 57, and Line 83.

Line 41 ff: Please rephrase the sentence, as computing platforms are not applied but used.

[Response]:

Thanks for the suggestion. We have replaced “applied” with “used” in the sentence at Line 42.

Line 55: Which gap do you precisely have in mind? Do you really mean climate science in general or rather climate modelling (aka Earth system modelling)?

[Response]:

Sorry for the confusion. The idea of gap or chasm is quoted from the reference (Bryan N. Lawrence et al, 2018, https://doi.org/10.5194/gmd-11-1799-2018). Due to the ever-increasing complexity of earth system models coupled with rapidly evolving computational techniques, climate modelling will arrive at a gap which will separate scientific aspiration from our ability to develop and/or rapidly adapt codes to the available hardware. We have replaced “climate science” with “climate modelling” at Line 42.

Line 70 ff: What is the former, what is the latter language? Could you briefly explain to the non-experts amongst the readers the difference between source-to-source and DSL? Perhaps this whole paragraph needs some restructuring (see remarks in my section 1.1).

[Response]:

Thanks for your comments. We will restructure this whole paragraph and add following sentences to explain the differences between source-to-source and DSL.

“Source-to-source translator can parse a piece of code as input and generate a novel code as output, such as from Fortran to C, or take a high-level language as input and convert the code with parallel code comments. While DSLs are developed to meet the growing needs of the development of applications. For example, Partitioned Global

Address Space (PGAS) takes the local memory in different processes as a global memory space and make network communication invisible to the programmers. Many advanced programing languages emerged from this concept, such as Coarray Fortran (Numrich et al, 1998; Mellor-Crummey et al., 2009), Unified Parallel C (El-Ghazawi and Smith, 2006), X10 (Charles et al., 2005), Chapel (Chamberlain et al., 2007) and XcalableMP (Nakao et al., 2012). Some other outstanding DSLs, such as OpenFoam (Jasak et al, 2007), ATMOL (Engelen, 2002), ICON DSL (Torres et al, 2013), and STELLA (Gysi et al, 2015), provide high-level abstraction interface close to the mathematical notations for domain scientists to write much more concise and simpler code."

Line 90: Please introduce the reader to the heterogeneity you have in mind here. What makes TaihuLight different in terms of heterogeneity? From the system specification further down in your text, TaihuLight does not look that heterogeneous.

[Response]:

Sorry for the confusion. One major technology innovation of the Sunway TaihuLight supercomputer is the homegrown SW26010 heterogeneous many-core processor. It includes four powerful core groups (CGs) connected via the network on chip (NoC), each of CGs consists of a manage processing element (MPE) and 64 computing processing elements (CPEs) arranged in an eight by eight mesh. We will add more details about the architecture of Sunway TaihuLight in the revised manuscript.

Line 100: I would say that you solved the problem for one ocean model or a particular class of ocean models but not yet for ocean models in general.

[Response]:

Thanks for your corrections. We have added "a particular class of" in front of "ocean models using the finite difference method and staggered grid in OpenArray." (Line 100)

Line 109: Is it really valid Fortran code, or shouldn't it better be classified as pseudo Fortran code as GOMO cannot really live without the OpenArray library?

[Response]:

Sorry for the confusion. OpenArray works as an independent library. As you pointed out, OpenArray is the essential for GOMO, since GOMO uses the functions and modules provided by OpenArray, such as average operators, differential operators, assignment functions, I/O functions, et al. While, GOMO is written by valid Fortran codes.

Line 111: Is it really meant like that you compile and link one executable which can then be executed on any computing platform? Or are you talking about the intermediate C++ code? But this would require compiling and linking on the target system before it can be executed.

[Response]:

Sorry for the confusion. As you mentioned, there is no free lunch in the world. We transfer the cross-platform complexity to OpenArray. When porting the ocean models

to a new platform, we need to redesign an additional function for the target system and add it to the code generation module in OpenArray. This added function is used to translate the intermediate computation graph into corresponding executable code for the target platform. OpenArray supports X86 and Sunway platforms, therefore GOMO is executable on the two platforms without additional modification.

We have changed the sentence "The final output is a program that is executable on different computing platforms." into "The final output is a program that is executable on the two computing platforms currently supported by OpenArray". (Line 111~112)

Line 120: True but only if the OpenArray has been ported to and is available on the Sunway platform. There is probably no free lunch when moving to a new hardware or software environment.

[Response]:

Thanks for your comments. As you mentioned, there is no free lunch when moving to a new hardware or software environment. In fact, we transfer the burden of porting code from GOMO to OpenArray, thus decoupling the ocean models from the hardware platforms is possible.

Line 148 and elsewhere: Equation 1 is probably taken from the POM user manual, but nevertheless should not expressions like $\frac{\partial \boldsymbol{DU}}{\partial \boldsymbol{x}}$ rather be written as $\frac{\partial}{\partial x}(\boldsymbol{DU})$?

[Response]:

Thanks for your suggestions. As you pointed out, the Eq. 1 is indeed derived from the POM user manual. Therefore, we would like to use the same expressions of the equations with that in POM user manual (shown as the following equation).

$$\frac{\partial DU}{\partial x} + \frac{\partial DV}{\partial y} + \frac{\partial \omega}{\partial \sigma} + \frac{\partial \eta}{\partial t} = 0$$

Line 152: Could provide some hint on how to arrive at the discrete expression (2).

[Response]:

Thanks for your suggestions. In this paragraph, we will provide the details of Arakawa C grid and finite difference method to demonstrate the process from Eq. (1) to the discrete Eq. (2).

In Arakawa C grid, fluid depth *D* is calculated at the centers. *U* component is calculated at the left and right side of the variable *D*, *V* component is calculated at the lower and upper side of the variable *D* (Shown as Fig. 1). Taking the term $\frac{\partial \boldsymbol{DU}}{\partial \boldsymbol{x}}$ for an example, we firstly apply linear interpolation to obtain the *D*'s value at *U* point represented by *tmp1*. Through a backward difference to the product of *tmp1* and *U*, then the discrete expression of $\frac{\partial \boldsymbol{DU}}{\partial \boldsymbol{x}}$ can be obtained.

tmp1 = 0.5(D(i+1,j)+D(i,j))*U(i+1,j)*

$$\frac{\partial DU}{\partial x} = \frac{0.5 * \big(D(i+1,j) + D(i,j)\big) * U(i+1,j) - 0.5 * \big(D(i,j) + D(i-1,j)\big) * U(i,j)}{dx(i,j)^*}$$

Where $dx(i,j)^* = 0.5*(dx(i,j) + dx(i-1,j))$

Unfortunately, we missed 0.5 in the Eq.2 (Line 153~154). We have changed the Eq.2 into the following form.

$$\frac{\eta_{t+1}(i,j) - \eta_{t-1}(i,j)}{2 * dt} + \frac{0.5 * \big(D(i+1,j) + D(i,j)\big) * U(i+1,j) - 0.5 * \big(D(i,j) + D(i-1,j)\big) * U(i,j)}{dx(i,j)^*}$$

$$+ \frac{0.5 * \big(D(i,j+1) + D(i,j)\big) * V(i,j+1) - 0.5 * \big(D(i,j) + D(i,j-1)\big) * V(i,j)}{dy(i,j)^*} = 0$$

Where $dx(i,j)^* = 0.5*(dx(i,j) + dx(i-1,j))$, $dy(i,j)^* = 0.5*(dy(i,j) + dy(i,j-1))$

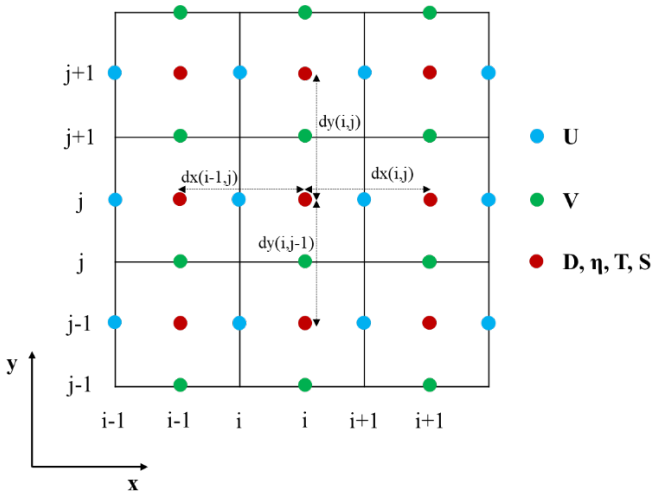

Figure 1. Arrangement of variables in the staggered Arakawa C grid.

Line 211: I thought your current implementation of the operators only supports uniform (=equidistant) grids? What am I missing here?

[Response]:

Sorry for the confusion. In OpenArray, grid increments (*dx*, *dy*, *dz*) are combined with physical variables through grid binding. On the staggering Arakawa C-grid, the components of velocity (*u*, *v*, *w*) and potential temperature (*T*) et al are defined at different points. Every variable on each point has its own set of grid increments so the current implementation of the operators supports varying grids.

Line 214 and elsewhere: I suggest to avoid the phrase “automatic”. Nothing is done automatically but every effect has a cause. Here, something is happening because you programmed it that way and some conditions are coming together to trigger an action.

[Response]:

Thanks for your corrections. We have replaced all the phrases “automatic/automatically” with “implicit/implicitly” in the revised manuscript. (Line 22, 29, 72, 74, 110, 214, 238,

268, 355)

Line 238 ff: Could you clarify how the Arakawa grid type, the jumping rules and the differential operators are linked together? Let us assume I have formulated my ocean model on an Arakawa C grid and now for curiosity would like to run it on an A grid (not because it would really makes sense but to demonstrate the effect of discretization on the numerical solution) what would I have to change in my ocean model code?

[Response]:

Thanks for your valuable suggestion. To make it clearer, we will take Eq. 1 as an example to demonstrate the effect of the numerical solution based on Arakawa A grid, B grid, and C grid and illustrate the modifications to the ocean models when grid scheme is changed in the revised manuscript.

Line 247 ff: What is the motivation for the list of ocean model codes you provide in this paragraph? There are other codes around, e.g. FESOM (see https://fesom.de and the list of publication there) or an unstructured grid model for global ocean dynamics by P. Korn (see https://doi.org/10.1016/j.jcp.2017.03.009) and several more.

[Response]:

Sorry for the confusion. In this paragraph, we want to express our respect to these excellent ocean models, such as POM, ROMS, MITgcm, FESOM et al. Considering most of these existing ocean models use finite difference or finite volume methods on structured meshes, we designed 12 basic operators in OpenArray only for this particular class of ocean models at present. In our future work, more customized operators will be implemented to support other numerical methods and meshes.

We have changed "…, including Advanced Circulation model (ADCIRC) …" into "…, including Finite-Element/volumE Sea ice-Ocean Mode (FESOM) (Korn, 2017), Advanced Circulation model (ADCIRC) …"

Line 274 section 3.1:

Could you add a few lines to describe the handling of lateral and vertical boundary conditions within your operators?

[Response]:

Thanks for your comments. In section 3.1, we will add the details to describe how to handle the lateral and vertical boundary conditions within the operators. First, the lateral and vertical boundary conditions are set to zeros within the operators and operator expressions. Then, GOMO invokes the modules which is independent with operators to update the boundary conditions.

Line 334: When speaking of subgraphs and the kernel function, can the individual advection and diffusion terms be accessed for diagnostic purposes?

[Response]:

Thanks for your comments. If users want access to individual variables or subgraphs, they need to split the formula or code into multiple expressions for diagnostic or

printing purposes.

Line 352: I am not sure what is meant here and perhaps the sentence should be rephrased. Such a function needs to be programmed once for a particular ocean model, but once it is there it can be used, see e.g. ESMF_FieldBundleHalo contained in the Earth System Modeling Framework (ESMF). To my knowledge, other ocean models use similar approaches for the halo exchange as well. But no doubt, it is a relief to have it.

[Response]:

Thanks for your suggestions, the sentence will be rephrased (Line 352). As you pointed out, earth system models generally call the particular functions to control the communication between the local boundary regions, such ESMF_FieldBundleHalo in ESMF. In OpenArray, we further hide these procedures of communication through the fused kernel. Users do not need to explicitly call these functions (e.g. ESMF_FieldBundleHalo) for the halo exchange, or know the parallel details.

Line 391: Is the mode splitting algorithm inherited from POM, if so, this should be mentioned.

[Response]:

Thanks for your suggestion. As you pointed out, the mode splitting algorithm is inherited from POM. We have changed "… the mode-splitting algorithm to address …" into "… the mode-splitting algorithm inherited from POM to address …" (Line 391)

Line 422: Could you provide the number of lines for OpenArray as well? I could calculate it myself but . . .

[Response]:

Thanks for your suggestion. The code of OpenArray is about 11,800 lines. We will add the number of lines for OpenArray into the sentence. (Line 422)

Line 425: You raise the impression that porting is not an issue anymore. While this is certainly true for GOMO (which is of course very valuable) the porting still has to be done for OpenArray. Maybe this could be clarified somewhere (perhaps in the discussion).

[Response]:

Thanks for your comments. As you pointed out, there is no free lunch when moving to a new hardware or software environment. Since the porting has been transferred to OpenArray, the models can be run on these platforms supported by OpenArray. We will clarify this in the discussion section in our revised manuscript.

Line 469 section 5.3: Which hardware do you use for these tests? What sets the upper bound of 4096 processes?

[Response]:

Thanks for your comments. We used the x86 cluster at National Supercomputing Center in Wuxi of China, which provides 5000 Intel Xeon E5-2650 v2 CPUs at most. In the future, we will increase the upper bound if more computing resources are available.

Line 481 section 5.4: What does this mean for a reasonable local domain size? 32X32 points as in sec. 5.3 is still on the good side, while 9x9 as used on TaihuLight does shows some performance degradation.
[Response]:
Thanks for your comments. On the same machine architecture, the parallel efficiency will be better if the local domain size is bigger. The architecture of Sunway TaihuLight is different from x86 machine architecture, so we did not compare and analyze the parallel scalability between them.

Line 490 ff: As I understand the steps up to compiling the JIT are done only once at the beginning of a job. If you run a longer experiment (in terms of wallclock time or number of timesteps) the initialisation phase should be negligible when compared to the total run time. Why don't you provide two numbers, one for the initialisation and one for the integration within the time loop?
[Response]:
Thanks for your suggestion. As you mentioned, the fusion-kernel codes are generated and compiled only once at the beginning of a job. In GOMO, this initialization phase consumes about 1 minute. In our revised manuscript, we will provide the consuming time for the initialization and the integration within the time loop, respectively. (Line 490)

3 Technical Corrections/Suggestions

L39 climate model → climate models
[Response]:
Corrected.

L42 climate community → climate modelling community
[Response]:
Corrected.

L43 model program needs → model programs need
[Response]:
Corrected.

L68 inefficiency → inefficiently
[Response]:
Corrected.

L83 the sentence needs to be reordered. It is not clear (to me) to which part "at the product level" is referring to.
[Response]:

Sorry for the confusion. To make a clear illustration, we have removed “at the product level”, and changed the sentence “…, one major difficulty is the requirement of a stable and robust compiler, rather than an experimental compiler, at the product level” into “…, one major difficulty is the requirement of a stable, reliable, and robust compiler with good support”. (Line 83~84)

L106 change to : . . . is similar to the original but manually optimized parallel program.
[Response]:
Corrected.

L108 support → supports
[Response]:
Corrected.

L125 → The implementation
[Response]:
Corrected.

L139 → In traditional ocean models . . .
[Response]:
Corrected.

L143 → When using the OpenArray library . . .
[Response]:
Corrected.

L211 → we propose
[Response]:
Corrected.

L220 → . . . the horizontal velocity components Array(U) and Array(V) are . . . or
. . . the horizontal velocity Array(U, V) is . . .
[Response]:
Thanks for your corrections. We have changed the sentence “. . . the horizontal velocity Array(U, V) are . . .” into “. . . the horizontal velocity Array(U, V) is . . .” (Line 220)

L238 can be used → are used
[Response]:
Corrected.

L257 different → difference ; operator → operators
[Response]:
Corrected.

L289 will be concealed by → is hidden behind
[Response]:
Corrected.

L290 → : : : and can escape . . .
[Response]:
Corrected.

L303 → to implement a so-called lazy expression . . .
[Response]:
Corrected.

L318 . . .AYF are the interpolated functions → . . .AYF are the average functions.
[Response]:
Corrected.

L373 computing processing elements: aren't these Central Processing Units (CPUs)
[Response]:
Sorry for the confusion.

In contrast with other existing heterogeneous supercomputers, which include both CPU processors and PCIe-connected many-core accelerators (NVIDIA GPU or Intel Xeon Phi), the computing power of TaihuLight is provided by a homegrown many-core SW26010 processor (or SW26010 CPU). The processor includes four core-groups (CGs). Each CG includes one management processing element (MPE), one computing processing element (CPE) cluster with eight by eight CPEs, and one memory controller (MC) (Shown as Fig. 2).

The MPE is a complete 64-bit RISC core, which can run in both the user and system modes. While, the CPE is also a 64-bit RISC core, but with limited functions. The CPE can only run in user mode and does not support interrupt functions. The CPE is designed to achieve the maximum aggregated computing power, while minimizing the complexity of the micro-architecture.

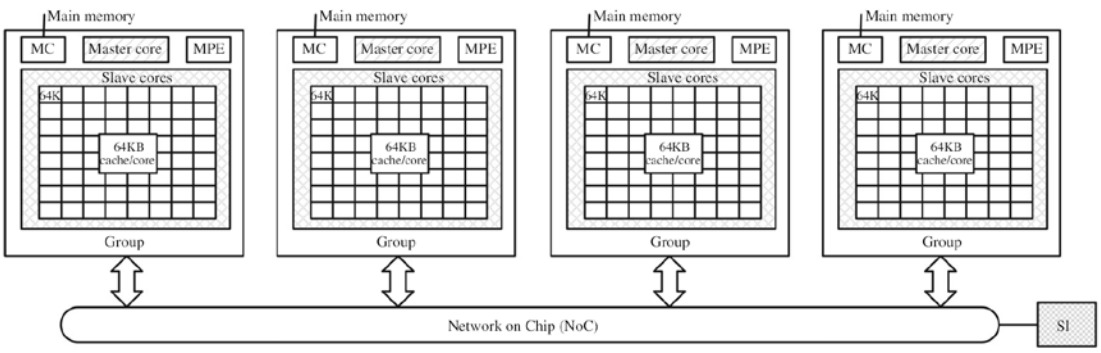

Figure 2. The architecture of the Sunway processor

Note: The paragraphs above and Fig. 2 are quoted from the reference (Haohuan Fu et al, 2016, https://link.springer.com/article/10.1007/s11432-016-5588-7).

To make a clear illustration, we will add more detailed introduction to the Sunway TaihuLight supercomputer in the revised manuscript.

L384 a practical ocean model → a numerical ocean model

[Response]:

Corrected.

L404 rephrase, the TKE and alike can be calculated but not the submodel.

[Response]:

Thanks for the suggestion. We have changed the sentence "…, and turbulence closure sub-model (q2, q2l) (Mellor and Yamada, 1982)" into "…, and turbulence closure scheme (q2, q2l) (Mellor and Yamada, 1982)".

L505 a practical ocean model → a numerical ocean model

[Response]:

Corrected.

We really appreciate your highly constructive comments.

Best wishes,

Xiaomeng Huang, Xing Huang, Dong Wang, Qi Wu et al.