

Dear editor and reviewer,

First of all, we would like to express our sincere appreciation to your efforts installing OpenArray and valuable feedbacks. Your comments are highly insightful and enable us to substantially improve the quality of OpenArray and the user manual.

Below are our point-to-point responses to all comments.

## 1 Responses to Referee #3

User Manual

Section 2 Installation

(1) Make it clear in the software requirements list, item 2 that you need only one of MPICH or openmpi or Intel MPI compiler.

**Response:**

Thanks for your helpful suggestions. Item2 has been modified to clarify the requirement of MPI library.

In section 2 Installation, We changed the original sentence “2) Message Passing Interface (MPI) compilers (MPICH v3.2.1 or later; Openmpi v3.0.0 or later; Intel MPI compiler 2017 or later).” into “2) A Message Passing Interface (MPI) library, MPICH (version 3.2.1 or later) or Openmpi (version 3.0.0 or later) or Intel MPI compiler (version 2017 or later).”

Section 2.1 installation on Linux

(2) Make it clear that you are assuming a "bash" shell, and the "export" commands will need to be altered for other shell environments.

**Response:**

We appreciate your valuable suggestions. To make it clear, we added an example for different shell environments in the section 2 and section 4 (shown as below).

In section 2.1 installation on Linux, we added

“The way to set environment variables is different for different types of shell, so you may need to use ‘setenv’ or ‘export’ or other command, an example is listed below:

```
setenv MPICC mpicc (for csh or tcsh)
export MPICC=mpicc (for sh, zsh or bash)
```

Below we are assuming the user is using sh, zsh, or bash.

”

In section 2.2 installation on Mac OS, we added:

“The way to set environment variables is different for different types of shell, so you may need to use ‘setenv’ or ‘export’ or other command, an example is listed below.

```
setenv MPICC mpicc (for csh or tcsh)
export MPICC=mpicc (for sh, zsh or bash)
```

Below we are assuming the user is using sh, zsh, or bash.

”

In section 4.2.2 installation, we added:

“For sh, zsh, or bash:

```
export OPENARRAY_DIR=${HOME}/install
export PNETCDF_DIR=${HOME}/install
```

For csh or tesh:

```
setenv OPENARRAY_DIR ${HOME}/install
setenv PNETCDF_DIR  ${HOME}/install
```

”

(3) The default installation directory for PnetCDF is actually /usr/local which may be fine for many users. The default installation directory for OpenArray is actually /usr/local (checked with ./configure --help).

**Response:**

Thanks for your efforts installing OpenArray and your comments. In the manual of OpenArray, we assume to install PnetCDF and OpenArray in *\${HOME}/install* since the default installation directory for PnetCDF and OpenArray (/usr/local) may cause permission issues for non-root users. To make it clearer, we revised the following sentences in the section 2.

In section 2.1 and 2.2, we added the following sentences: “Note: we are assuming *\${HOME}/install* as the installation path of PnetCDF and OpenArray, since the default */usr/local* directory may cause permission issues for non-root users. However, the users can change it by specifying a different *PREFIX* value.”

In section 2.1 item (b), we changed the original sentences “The default installation directory of PnetCDF is *\${HOME}/install*” into “The default installation directory of PnetCDF is */usr/local* which might cause permission issues for non-root users. Thus we are assuming to install PnetCDF in *\${HOME}/install*.”

In section 2.1 item (c), we changed the original sentences “The default installation directory of OpenArray is *\${HOME}/install*.” into “Assuming the installation directory of OpenArray is *\${HOME}/install*.”

In section 2.2 item (c), we changed the original sentences “The installation directory of PnetCDF is *\${HOME}/install*.” into “The default installation directory of PnetCDF is */usr/local* which might cause permission issues for non-root users. Thus we are assuming to install PnetCDF in *\${HOME}/install*.”

In section 2.2 item (d), we changed the original sentences “The default installation directory of OpenArray is *\${HOME}/install*.” into “Assuming the installation directory of OpenArray is *\${HOME}/install*.”

(4) You should give example output from *./manual\_main* (at least a few lines) for the user to compare to.

**Response:**

Thanks for your helpful suggestions. We added the example outputs from `./manual_main` and `./GOMO` in the manual shown as the following figures.

```
[huangxing@earth OpenArray-master]$ ./manual_main
zeros =
  data type = float
  pos = -1
  is_pseudo = 0
  bitset = 111
  global_shape = [2, 2, 2]
  procs_shape = [1, 1, 1]
  bound_type = [0, 0, 0]
  stencil_type = 1
  stencil_width = 1
  lx = [2]
  ly = [2]
  lz = [2]
  clx = [0, 2]
  cly = [0, 2]
  clz = [0, 2]
[k = 0]
  j = 0          j = 1
i = 0  0.0000000000000000  0.0000000000000000
i = 1  0.0000000000000000  0.0000000000000000
[k = 1]
  j = 0          j = 1
i = 0  0.0000000000000000  0.0000000000000000
i = 1  0.0000000000000000  0.0000000000000000
ones =
  data type = float
  pos = -1
  is_pseudo = 0
  bitset = 111
  global_shape = [2, 2, 2]
  procs_shape = [1, 1, 1]
  bound_type = [0, 0, 0]
  stencil_type = 1
  stencil_width = 1
  lx = [2]
  ly = [2]
  lz = [2]
  clx = [0, 2]
  cly = [0, 2]
  clz = [0, 2]
[k = 0]
  j = 0          j = 1
i = 0  1.0000000000000000  1.0000000000000000
i = 1  1.0000000000000000  1.0000000000000000
[k = 1]
  j = 0          j = 1
i = 0  1.0000000000000000  1.0000000000000000
i = 1  1.0000000000000000  1.0000000000000000
```

Figure 1. The first few lines of the output from `./manual_main`.

```

[huangxing@earth bin]$ ./GOMO
start reading init variables...
nsbdy = 1 iint= 1
vamax = 5.0333135650657786E-002
*****
time= 0.0020833; iint= 1; iext= 31; iprint= 6
*****
vtot= 393292026336165.0625; atot= 89167177231.1113892; eaver= -0.0006871
taver= 8.3401626508; saver= 35.0000000000; tsalt= 13765220921765788.0000000
nsbdy = 1 iint= 2
vamax = 4.0942026145322240E-002
*****
time= 0.0041667; iint= 2; iext= 31; iprint= 6
*****
vtot= 393291977119409.3750; atot= 89167177231.1113892; eaver= -0.0012391
taver= 8.3401626465; saver= 35.0000000000; tsalt= 13765219199183138.0000000
nsbdy = 1 iint= 3
vamax = 4.7123633123959043E-002
*****
time= 0.0062500; iint= 3; iext= 31; iprint= 6
*****
vtot= 393291928442704.7500; atot= 89167177231.1113892; eaver= -0.0017850
taver= 8.3401626547; saver= 35.0000000000; tsalt= 13765217495563576.0000000
nsbdy = 1 iint= 4
vamax = 4.6187087118355202E-002
*****
time= 0.0083333; iint= 4; iext= 31; iprint= 6
*****
vtot= 3932918271124303.8750; atot= 89167177231.1113892; eaver= -0.0029213
taver= 8.3401627246; saver= 35.0000000001; tsalt= 13765213949383962.0000000
nsbdy = 1 iint= 5
vamax = 4.3295450442905815E-002
*****
time= 0.0104167; iint= 5; iext= 31; iprint= 6
*****
vtot= 393291637375164.0000; atot= 89167177231.1113892; eaver= -0.0050493
taver= 8.3401626654; saver= 35.0000000007; tsalt= 13765207308421706.0000000

```

Figure 2. The first few lines of the output from *./GOMO*.

## 2 Changes in the manuscript

(1) In the acknowledgements section, we changed the sentence “Xiaomeng Huang is supported by a grant from the State’s Key Project of Research and Development Plan (2016YFB0201100) and the National Natural Science Foundation of China (41776010).” into “Xiaomeng Huang is supported by a grant from the State’s Key Project of Research and Development Plan (2016YFB0201100), the National Natural Science Foundation of China (41776010), and Center for High Performance Computing and System Simulation of Pilot National Laboratory for Marine Science and Technology (Qingdao)” (Lines 768~772)

(2) In the figure section, we modified the Fig.1 (shown as below) and changed the font in Fig.1~9 and Fig.12~13.

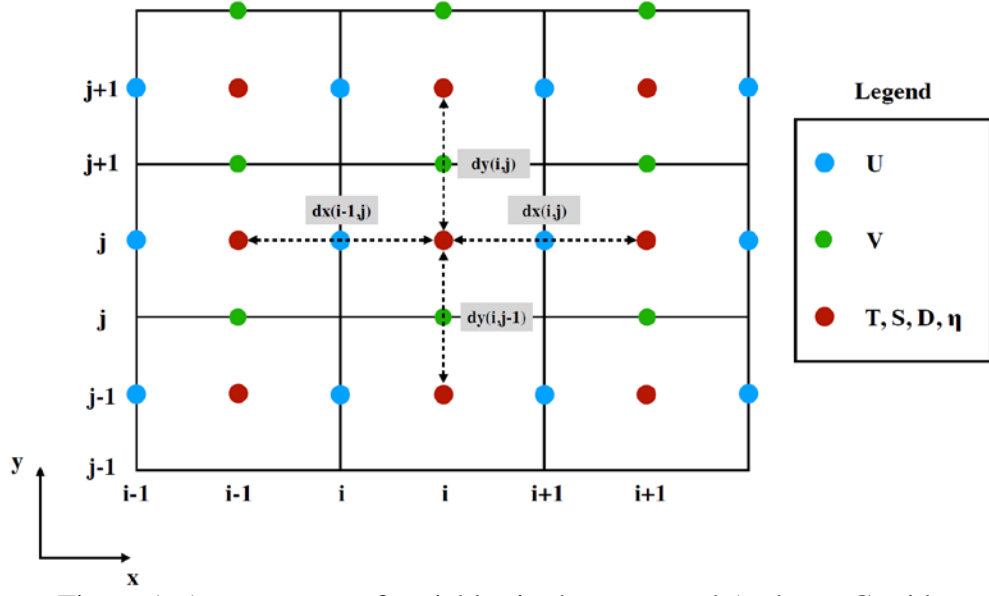


Figure 1. Arrangement of variables in the staggered Arakawa C grid.

# 1 **OpenArray v1.0: A Simple Operator Library for the Decoupling of**

## 2 **Ocean Modelling and Parallel Computing**

3  
4 Xiaomeng Huang<sup>1,2,3</sup>, Xing Huang<sup>1,3</sup>, Dong Wang<sup>1,3</sup>, Qi Wu<sup>1</sup>, Yi Li<sup>3</sup>, Shixun Zhang<sup>3</sup>,  
5 Yuwen Chen<sup>1</sup>, Mingqing Wang<sup>1,3</sup>, Yuan Gao<sup>1</sup>, Qiang Tang<sup>1</sup>, Yue Chen<sup>1</sup>, Zheng Fang<sup>1</sup>,  
6 Zhenya Song<sup>2,4</sup>, Guangwen Yang<sup>1,3</sup>

7  
8 <sup>1</sup> Ministry of Education Key Laboratory for Earth System Modeling, Department of  
9 Earth System Science, Tsinghua University, Beijing 100084, China

10 <sup>2</sup> Laboratory for Regional Oceanography and Numerical Modeling, Qingdao National  
11 Laboratory for Marine Science and Technology, Qingdao, 266237, China

12 <sup>3</sup> National Supercomputing Center in Wuxi, Wuxi, 214011, China

13 <sup>4</sup> First Institute of Oceanography, Ministry of Natural Resources, Qingdao, 266061,  
14 China

15  
16 Corresponding author: [hxm@tsinghua.edu.cn](mailto:hxm@tsinghua.edu.cn)

### 17 **Abstract**

18 The rapidly evolving computational techniques are making a large gap between  
19 scientific aspiration and code implementation in climate modelling. In this work, we  
20 design a simple computing library to bridge the gap and decouple the work of ocean  
21 modelling from parallel computing. This library provides twelve basic operators that  
22 feature user-friendly interfaces, effective programming and implicit parallelism.  
23 Several state-of-art computing techniques, including computing graph and Just-In-Time  
24 compiling are employed to parallelize the seemingly serial code and speed up the ocean  
25 models. These operator interfaces are designed using native Fortran programming  
26 language to smooth the learning curve. We further implement a highly readable and  
27 efficient ocean model that contains only 1860 lines of code but achieves a 91% parallel  
28 efficiency in strong scaling and 99% parallel efficiency in weak scaling with 4096 Intel  
29 CPU cores. This ocean model also exhibits excellent scalability on the heterogeneous

30 Sunway TaihuLight supercomputer. This work presents a promising alternative tool for  
31 the development of ocean models.

32

33 **Keywords:** implicit parallelism, operator, ocean modelling, parallel computing

## 34 **1. Introduction**

35 Many earth system models have been developed in the past several decades to improve  
36 the predictive understanding of the earth system (Bonan and Doney, 2018; Collins et  
37 al., 2018; Taylor et al., 2012). These models are becoming increasingly complicated,  
38 and the amount of code has expanded from a few thousand lines to tens of thousands,  
39 or even millions of lines. In terms of software engineering, an increase in code causes  
40 the models to be more difficult to develop and maintain.

41

42 The complexity of these models mainly originates from three aspects. First, more model  
43 components and physical processes have been embedded into the earth system models,  
44 leading to a tenfold increase in the amount of code (e.g., Alexander and Easterbrook,  
45 2015). Second, some heterogeneous and advanced computing platforms (e.g.,  
46 Lawrence et al., 2018) have been widely used by the climate modelling community,  
47 resulting in a fivefold increase in the amount of code (e.g., Xu et al., 2015). Last, most  
48 of the model programs need to be rewritten due to the continual development of novel  
49 numerical methods and meshes. The promotion of novel numerical methods and  
50 technologies produced in the fields of computational mathematics and computer  
51 science have been limited in climate science because of the extremely heavy burden  
52 caused by program rewriting and migration.

53

54 Over the next few decades, tremendous computing capacities will be accompanied by  
55 more heterogeneous architectures which are equipped with two or more kinds of cores  
56 or processing elements (Shan, 2006), thus making for a much more sophisticated  
57 computing environment for climate modellers than ever before (Bretherton et al., 2012).  
58 Clearly, transiting the current earth system models to the next generation of computing  
59 environments will be highly challenging and disruptive. Overall, complex codes in  
60 earth system models combined with rapidly evolving computational techniques create  
61 a very large gap between scientific aspiration and code implementation in the climate  
62 modelling community.



63

64 To reduce the complexity of earth system models and bridge this gap, a universal and  
65 productive computing library is a promising solution. Through establishing an implicit  
66 parallel and platform-independent computing library, the complex models can be  
67 simplified and will no longer need explicit parallelism and transiting, thus effectively  
68 decoupling the development of ocean models from complicated parallel computing  
69 techniques and diverse heterogeneous computing platforms.

70

71 Many efforts have been made to address the complexity of parallel programming for  
72 numerical simulations, such as operator overloading, source-to-source translator and  
73 domain specific language (DSL). Operator overloading supports the customized data  
74 type and provides simple operators and function interfaces to implement the model  
75 algorithm. This technique is widely used because the implementation is straightforward  
76 and easy to understand (Corliss and Griewank, 1994; Walther et al., 2003). However, it  
77 is prone to work inefficiently because overloading execution induces numerous  
78 unnecessary intermediate variables, consuming valuable memory bandwidth resources.  
79 Using a source-to-source translator offers another solution. As indicated by the name,  
80 this method converts one language, which is usually strictly constrained by self-defined  
81 rules, to another (Bae et al., 2013; Lidman et al., 2012). It requires tremendous work to  
82 develop and maintain a robust source-to-source compiler. Furthermore, DSLs can  
83 provide high-level abstraction interfaces that use mathematical notations similar to  
84 those used by domain scientists, so that they can write much more concise and more  
85 straightforward code. Some outstanding DSLs, such as ATMOL (van Engelen, 2001),  
86 ICON DSL (Torres et al., 2013), STELLA (Gysi et al., 2015) and ATLAS (Deconinck  
87 et al., 2017), are used by the numerical model community. Although they seem source-  
88 to-source technique, DSLs are newly-defined languages and produce executable  
89 programs instead of target languages. Therefore the new syntax makes it difficult for  
90 the modellers to master the DSLs. In addition, most DSLs are not yet supported by  
91 robust compilers due to their relatively short history. Most of the source-to-source

92 translators and DSLs still do not support the rapidly evolving heterogeneous computing  
93 platforms, such as the Chinese Sunway TaihuLight supercomputer which is based on  
94 the homegrown Sunway heterogeneous many-core processors and located at the  
95 National Supercomputing Center in Wuxi.

96

97 Other methods such as COARRAY Fortran and CPP templates provide alternative ways.  
98 Using COARRAY Fortran, a modeller has to control the reading and writing operation  
99 of each image (Mellor-Crummey et al., 2009). In a sense, one has to manipulate the  
100 images in parallel instead of writing serial code. In term of CPP templates, it is usually  
101 suitable for small code and difficult for debugging (Porkoláb et al., 2007).

102

103 Inspired by the philosophy of operator overloading, source-to-source translating and  
104 DSLs, we integrated the advantages of these three methods into a simple computing  
105 library which is called OpenArray. The main contributions of OpenArray are as follows:

- 106 • Easy-to-use. The modellers can write simple operator expressions in Fortran to  
107 solve partial differential equations (PDEs). The entire program appears to be  
108 serial and the modellers do not need to know any parallel computing techniques.  
109 We summarized twelve basic generalized operators to support whole  
110 calculations in a particular class of ocean models which use the finite difference  
111 method and staggered grid.
- 112 • High efficiency. We adopt some advanced techniques, including intermediate  
113 computation graphing, asynchronous communication, kernel fusion, loop  
114 optimization, and vectorization, to decrease the consumption of memory  
115 bandwidth and improve efficiency. Performance of the programs implemented  
116 by OpenArray is similar to the original but manually optimized parallel program.
- 117 • Portability. Currently OpenArray supports both CPU and Sunway platforms.  
118 More platforms including GPU will be supported in the future. The complexity  
119 of cross-platform migration is moved from the models to OpenArray. The  
120 applications based on OpenArray can then be migrated seamlessly to the

121 supported platforms.

122

123 Furthermore, we developed a numerical ocean model based on the Princeton Ocean  
124 Model (POM, Blumberg and Mellor, 1987) to test the capability and efficiency of  
125 OpenArray. The new model is called the Generalized Operator Model of the Ocean  
126 (GOMO). Because the parallel computing details are completely hidden, GOMO  
127 consists of only 1860 lines of Fortran code and is more easily understood and  
128 maintained than the original POM. Moreover, GOMO exhibits excellent scalability and  
129 portability on both central processing unit (CPU) and Sunway platforms.

130

131 The remainder of this paper is organized as follows. Section 2 introduces some concepts  
132 and presents the detailed mathematical descriptions of formulating the PDEs into  
133 operator expressions. Section 3 describes the detailed design and optimization  
134 techniques of OpenArray. The implementation of GOMO is described in section 4.  
135 Section 5 evaluates the performances of OpenArray and GOMO. Finally, discussion  
136 and conclusion are given in section 6 and 7, respectively.

137

## 138 **2. Concepts of the Array, Operator, and Abstract Staggered Grid**

139 In this section, we introduce three important concepts in OpenArray: Array, Operator  
140 and Abstract Staggered Grid to illustrate the design of OpenArray.

141

### 142 **2.1 Array**

143 To achieve this simplicity, we designed a derived data type, *Array*, which inspired our  
144 project name, OpenArray. The new *Array* data type comprises a series of information,  
145 including a 3-dimensional (3D) array to store data, a pointer to the computational grid,  
146 a Message Passing Interface (MPI) communicator, the size of the halo region and other  
147 information about the data distribution. All the information is used to manipulate the  
148 *Array* as an object to simplify the parallel computing. In traditional ocean models,  
149 calculations for each grid point and the  $i$ ,  $j$ , and  $k$  loops in the horizontal and vertical

150 directions are unavoidable. The advantage of taking the *Array* as an object is the  
 151 significant reduction in the number of loop operations in the models, making the code  
 152 more intuitive and readable. When using the OpenArray library in a program, one can  
 153 use *type(Array)* to declare new variables.

154

## 155 2.2 Operator

156 To illustrate the concept of an operator, we first take a 2-dimensional (2D) continuous  
 157 equation solving sea surface elevation as an example:

$$158 \quad \frac{\partial \eta}{\partial t} + \frac{\partial DU}{\partial x} + \frac{\partial DV}{\partial y} = 0, \quad (1)$$

159 where  $\eta$  is the surface elevation,  $U$  and  $V$  are the zonal and meridional velocities, and  
 160  $D$  is the depth of the fluid column. We choose the finite difference method and staggered  
 161 Arakawa C grid scheme, which are adopted by most regional ocean models. In Arakawa  
 162 C grid,  $D$  is calculated at the centers,  $U$  component is calculated at the left and right  
 163 side of the variable  $D$ ,  $V$  component is calculated at the lower and upper side of the  
 164 variable  $D$  (Fig. 1). Variables ( $D$ ,  $U$ ,  $V$ ) located at different positions own different sets  
 165 of grid increments. Taking the term  $\frac{\partial DU}{\partial x}$  as an example, we firstly apply linear  
 166 interpolation to obtain the  $D$ 's value at  $U$  point represented by  $tmpD$ . Through a  
 167 backward difference to the product of  $tmpD$  and  $U$ , then the discrete expression of  $\frac{\partial DU}{\partial x}$   
 168 can be obtained.

$$169 \quad tmpD(i+1,j) = 0.5*(D(i+1,j)+D(i,j))*U(i+1,j), \quad (2)$$

170 and

$$171 \quad \frac{\partial DU}{\partial x} = \frac{tmpD(i+1,j)-tmpD(i,j)}{dx(i,j)^*} = \frac{0.5*(D(i+1,j)+D(i,j))*U(i+1,j)-0.5*(D(i,j)+D(i-1,j))*U(i,j)}{dx(i,j)^*}, \quad (3)$$

172 where  $dx(i,j)^* = 0.5*(dx(i,j) + dx(i-1,j))$ .

173

174 In this way, the above continuous equation can be discretized into the following form.

$$175 \quad \frac{\eta_{t+1}(i,j)-\eta_{t-1}(i,j)}{2*dt} + \frac{0.5*(D(i+1,j)+D(i,j))*U(i+1,j)-0.5*(D(i,j)+D(i-1,j))*U(i,j)}{dx(i,j)^*} +$$

$$176 \quad \frac{0.5*(D(i,j+1)+D(i,j))*V(i,j+1)-0.5*(D(i,j)+D(i,j-1))*V(i,j)}{dy(i,j)^*} = 0, \quad (4)$$

177 where  $dx(i,j)^* = 0.5 * ( dx(i,j) + dx(i-1,j) )$ ,  $dy(i,j)^* = 0.5 * ( dy(i,j) + dy(i,j-1) )$ ,  
 178 subscripts  $\eta_{t+1}$  and  $\eta_{t-1}$  denote the surface elevations at the  $(t+1)$  time step and  $(t-1)$  time  
 179 step. To simplify the discrete form, we introduce some notation for the differentiation  
 180  $(\delta_f^x, \delta_b^y)$  and interpolation  $(\overline{\quad}_f^x, \overline{\quad}_b^y)$ . The  $\delta$  and overbar symbols define the  
 181 differential operator and average operator. The subscript  $x$  or  $y$  denotes that the  
 182 operation acts in the  $x$  or  $y$  direction, and the superscript  $f$  or  $b$  denotes that the  
 183 approximation operation is forward or backward.

184

185 Table 1 lists the detailed definitions of the twelve basic operators. The term *var* denotes  
 186 a 3D model variable. All twelve operators for the finite difference calculations are  
 187 named using three letters in the form [A|D][X|Y|Z][F|B]. The first letter contains two  
 188 options, A or D, indicating an average or a differential operator. The second letter  
 189 contains three options, X, Y or Z, representing the direction of the operation. The last  
 190 letter contains two options, F or B, representing forward or backward operation. The  
 191  $dx$ ,  $dy$  and  $dz$  are the distances between two adjacent grid points along the  $x$ ,  $y$  and  $z$   
 192 directions.

193 Using the basic operators, Eq. (4) is expressed as:

$$194 \quad \frac{\eta_{t+1} - \eta_{t-1}}{2 * dt} + \delta_f^x (\overline{D}_b^x * U) + \delta_f^y (\overline{D}_b^y * V) = 0. \quad (5)$$

195 Thus,

$$196 \quad \eta_{t+1} = \eta_{t-1} - 2 * dt * \left( \delta_f^x (\overline{D}_b^x * U) + \delta_f^y (\overline{D}_b^y * V) \right). \quad (6)$$

197 Then, Eq. (6) can be easily translated into a line of code using operators (the bottom  
 198 left panel in Fig. 2). Compared with the pseudo-codes (the right panel), the  
 199 corresponding implementation by operators is more straightforward and more  
 200 consistent with the equations.

201

202 Next, we will use the operators in shallow water equations, which are more complicated  
 203 than those in the previous case. Assuming that the flow is in hydrostatic balance and  
 204 that the density and viscosity coefficients are constant, and neglecting the molecular  
 205 friction, the shallow water equations are:

$$206 \quad \frac{\partial \eta}{\partial t} + \frac{\partial DU}{\partial x} + \frac{\partial DV}{\partial y} = 0, \quad (7)$$

$$207 \quad \frac{\partial DU}{\partial t} + \frac{\partial DUU}{\partial x} + \frac{\partial DVU}{\partial y} - fVD = -gD \frac{\partial \eta}{\partial x} + \mu D \left( \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} \right), \quad (8)$$

$$208 \quad \frac{\partial DV}{\partial t} + \frac{\partial DUV}{\partial x} + \frac{\partial DVV}{\partial y} + fUD = -gD \frac{\partial \eta}{\partial y} + \mu D \left( \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} \right), \quad (9)$$

209 where  $f$  is the Coriolis parameter,  $g$  is the gravitational acceleration, and  $\mu$  is the  
 210 coefficient of kinematic viscosity. Using the Arakawa C grid and leapfrog time  
 211 difference scheme, the discrete forms represented by operators are shown in Eq. (10) ~  
 212 Eq. (12).

$$213 \quad \frac{\eta_{t+1} - \eta_{t-1}}{2*dt} + \delta_f^x (\bar{D}_b^x * U) + \delta_f^y (\bar{D}_b^y * V) = 0, \quad (10)$$

$$214 \quad \frac{D_{t+1}U_{t+1} - D_{t-1}U_{t-1}}{2*dt} + \delta_b^x (\overline{D_b^x * U}_f^x * \bar{U}_f^x) + \delta_f^y (\overline{D_b^y * V}_b^x * \bar{U}_b^y) - \overline{f \bar{V}_f^y * D}_b^x = -g *$$

$$215 \quad \bar{D}_b^x * \delta_b^x(\eta) + \mu * \bar{D}_b^x * \left( \delta_b^x (\delta_f^x (U_{t-1})) + \delta_f^y (\delta_b^y (U_{t-1})) \right), \quad (11)$$

$$216 \quad \frac{D_{t+1}V_{t+1} - D_{t-1}V_{t-1}}{2*dt} + \delta_f^x (\overline{D_b^x * U}_b^y * \bar{V}_b^x) + \delta_b^y (\overline{D_b^y * V}_f^y * \bar{V}_f^y) + \overline{f \bar{U}_f^x * D}_b^y = -g *$$

$$217 \quad \bar{D}_b^y * \delta_b^y(\eta) + \mu * \bar{D}_b^y * \left( \delta_f^x (\delta_b^x (V_{t-1})) + \delta_b^y (\delta_f^y (V_{t-1})) \right). \quad (12)$$

218 As the shallow water equations are solved, spatial average and differential operations  
 219 are called repeatedly. Implementing these operations is troublesome and thus it is  
 220 favourable to abstract these common operations from PDEs and encapsulate them into  
 221 user-friendly, platform-independent, and implicit parallel operators. As shown in Fig.  
 222 3, we require only 3 lines of code to solve the shallow water equations. This more  
 223 realistic case suggests that even more complex PDEs can be constructed and solved by  
 224 following this elegant approach.

225

### 226 **2.3 Abstract staggered grid**

227 Most ocean models are implemented based on the staggered Arakawa grids (Arakawa  
 228 and Lamb, 1981; Griffies et al., 2000). The variables in ocean models are allocated at  
 229 different grid points. The calculations that use these variables are performed after  
 230 several reasonable interpolations or differences. When we call the differential  
 231 operations on a staggered grid, the difference value between adjacent points should be  
 232 divided by the grid increment to obtain the final result. Setting the correct grid  
 233 increment for modellers is troublesome work that is extremely prone to error, especially

234 when the grid is nonuniform. Therefore, we propose an abstract staggered grid to  
235 support flexible switching of operator calculations among different staggered grids.  
236 When the grid information is provided at the initialization phase of OpenArray, a set of  
237 grid increments, including horizontal increments ( $dx(i,j)$ ,  $dy(i,j)$ ) and vertical increment  
238 ( $dz(k)$ ), will be combined with each corresponding physical variable through grid  
239 binding. Thus, the operators can implicitly set the correct grid increments for different  
240 *Array* variables, even if the grid is nonuniform.

241

242 As shown in Fig. 4, the cubes in the (a), (b), (c), and (d) panels are the minimum abstract  
243 grid accounting for 1/8 of the volume of the cube in Panel (e). The eight points of each  
244 cube are numbered sequentially from 0 to 7, and each point has a set of grid increments,  
245 i.e.,  $dx$ ,  $dy$  and  $dz$ . For example, all the variables of an abstract Arakawa A grid are  
246 located at Point 3. For the Arakawa B grid, the horizontal velocity *Array* ( $U$ ,  $V$ ) is  
247 located at Point 0, the temperature ( $T$ ), the salinity ( $S$ ), and the depth ( $D$ ) are located at  
248 Point 3, and the vertical velocity *Array* ( $W$ ) is located at Point 7. For the Arakawa C  
249 grid, *Array U* is located at Point 2 and *Array V* is located at Point 1. In contrast, for the  
250 Arakawa D grid, *Array U* is located at Point 1 and *Array V* is located at Point 2.

251

252 When we call the average and differential operators mentioned in Table 1, for example,  
253 on the abstract Arakawa C grid, the position of *Array D* is Point 3, and the average  $AXB$   
254 operator acting on *Array D* will change the position from Point 3 to Point 1. Since *Array*  
255  $U$  is also allocated at Point 1, the operation  $AXB(D)*U$  is allowed. In addition, the  
256 subsequent differential operator on *Array*  $AXB(D)*U$  will change the position of *Array*  
257  $DXF(AXB(D)*U)$  from Point 1 to Point 3.

258

259 The jumping rules of different operators are given in Table 2. Due to the design of the  
260 abstract staggered grids, the jumping rules for the Arakawa A, B, C, and D grids are  
261 fixed. A change in the position of an array is determined only by the direction of a  
262 certain operator acting on that array.

263

264 If users change the Arakawa grid type, first the position information of each physical  
 265 variable need to be reset (Shown in Fig. 4). Then the discrete form of each equation  
 266 needs to be redesigned. We take the Eq. (1) switching from Arakawa C grid to Arakawa  
 267 B grid as an example. The positions of the horizontal velocity *Array U* and *Array V* are  
 268 changed to Point 0, *Array η* and *Array D* stay the same. The discrete form is changed  
 269 from Eq. (4) to Eq. (13), the corresponding implementation by operators is changed  
 270 from Eq. (6) to Eq. (14).

$$\begin{aligned}
 271 \quad & \frac{\eta_{t+1}(i,j) - \eta_{t-1}(i,j)}{2*dt} + \frac{0.25*(D(i+1,j)+D(i,j))*(U(i+1,j)+U(i+1,j+1)) - 0.25*(D(i,j)+D(i-1,j))*(U(i,j)+U(i,j+1))}{dx(i,j)*} + \\
 272 \quad & \frac{0.25*(D(i,j+1)+D(i,j))*(V(i,j+1)+V(i+1,j+1)) - 0.25*(D(i,j)+D(i,j-1))*(V(i,j)+V(i+1,j))}{dy(i,j)*} = 0, \\
 273 \quad & \hspace{15em} (13)
 \end{aligned}$$

$$274 \quad \eta_{t+1} = \eta_{t-1} - 2 * dt * \left( \delta_f^x \left( \bar{D}_b^x * \bar{U}_f^y \right) + \delta_f^y \left( \bar{D}_b^y * \bar{V}_f^x \right) \right). \quad (14)$$

275 The position information and jumping rules are used to implicitly check whether the  
 276 discrete form of an equation is correct. The grid increments are hidden by all the  
 277 differential operators, thus it makes the code simple and clean. In addition, since the  
 278 rules are suitable for multiple staggered Arakawa grids, the modellers can flexibly  
 279 switch the ocean model between different Arakawa grids. Notably, the users of  
 280 OpenArray should input the correct positions of each array in the initialization phase.  
 281 The value of the position is an input parameter when declaring an *Array*. An error will  
 282 be reported if an operation is performed between misplaced points.

283

284 Although most of the existing ocean models use finite difference or finite volume  
 285 methods on structured or semi-structured meshes (e.g., Blumberg and Mellor, 1987;  
 286 Shchepetkin and McWilliams, 2005), there are still some ocean models using  
 287 unstructured meshes (e.g., Chen et al., 2003; Korn, 2017), and even the spectral element  
 288 method (e.g., Levin et al., 2000). In our current work, we design the basic operators  
 289 only for finite difference and finite volume methods with structured grids. More  
 290 customized operators for the other numerical methods and meshes will be implemented



291 in our future work.

292

### 293 **3. Design of OpenArray**

294 Through the above operator notations in Table 1, ocean modellers can quickly convert  
295 the discrete PDE equations into the corresponding operator expression forms. The main  
296 purpose of OpenArray is to make complex parallel programming transparent to the  
297 modellers. As illustrated in Fig. 5, we use a computation graph as an intermediate  
298 representation, meaning that the operator expression forms written in Fortran will be  
299 translated into a computation graph with a particular data structure. In addition,  
300 OpenArray will use the intermediate computation graph to analyse the dependency of  
301 the distributed data and produce the underlying parallel code. Finally, we use stable and  
302 mature compilers, such as the GNU Compiler Collection (GCC), Intel compiler (ICC),  
303 and Sunway compiler (SWACC), to generate the executable programs according to  
304 different backend platforms. These four steps and some related techniques are described  
305 in detail in this section.

306

#### 307 **3.1 Operator expression**

308 Although the basic generalized operators listed in Table 1 are only suitable to execute  
309 first-order difference, other high-order difference or even more complicated operations  
310 can be combined by these basic operators. For example, a second-order difference  
311 operation can be expressed as  $\delta_f^x(\delta_b^x(var))$ . Supposing the grid distance is uniform,  
312 the corresponding discrete form is  $[var(i+1,j,k)+var(i-1,j,k) -2* var(i,j,k) ] / dx^2$ . In  
313 addition, the central difference operation can be expressed as  $(\delta_f^x(var) + \delta_b^x(var))/2$   
314 since the corresponding discrete form is  $[var(i+1,j,k)-var(i-1,j,k)] / 2dx$ .

315

316 Using these operators to express the discrete PDE equation, the code and formula are  
317 very similar. We call this effect “the self-documenting code is the formula”. Fig. 6  
318 shows the one-to-one correspondence of each item in the code and the items in the sea  
319 surface elevation equation. The code is very easy to program and understand. Clearly,

320 the basic operators and the combined operators greatly simplify the development and  
321 maintenance of ocean models. The complicated parallel and optimization techniques  
322 are hidden behind these operators. Modellers no longer need to care about details and  
323 can escape from the “parallelism swamp”, and can therefore concentrate on the  
324 scientific issues.

325

### 326 **3.2 Intermediate computation graph**

327 Considering the example mentioned in Fig. 6, if one needs to compute the term  
328  $DXF(AXB(D)*u)$  with the traditional operator overloading method, one first computes  
329  $AXB(D)$  and stores the result into a temporary array (named  $tmp1$ ), and then executes  
330  $(tmp1*u)$  and stores the result into a new array,  $tmp2$ . The last step is to compute  
331  $DXF(tmp2)$  and store the result in a new array,  $tmp3$ . Numerous temporary arrays  
332 consume a considerable amount of memory, making the efficiency of operator  
333 overloading is poor.

334

335 To solve this problem, we convert an operator expression form into a directed and  
336 acyclic graph, which consists of basic data and function nodes, to implement a so-called  
337 lazy expression evaluation (Bloss et al., 1988; Reynolds, 1999). Unlike the traditional  
338 operator overloading method, we overload all arithmetic functions to generate an  
339 intermediate computation graph rather than to obtain the result of each function. This  
340 method is widely used in deep learning frameworks, e.g., TensorFlow (Abadi et al.,  
341 2016) and Theano (Bastien et al., 2012), to improve computing efficiency. Figure 7  
342 shows the procedure of parsing the operator expression form of the sea level elevation  
343 equation into a computation graph. The input variables in the square boxes include the  
344 sea surface elevation ( $elb$ ), the zonal velocity ( $u$ ), the meridional velocity ( $v$ ) and the  
345 depth ( $D$ ).  $dt2$  is a constant equal to  $2*dt$ . The final output is the sea surface elevation  
346 at the next time step ( $elf$ ). The operators in the round boxes have been overloaded in  
347 OpenArray. In summary, all the operators provided by OpenArray are functions for the  
348 Array calculation, in which the “=” notation is the assignment function, the “-” notation  
349 is the subtraction function, the “\*” notation is the multiplication function, the “+”

350 notation is the addition function, DXF and DYF are the differential functions, and AXF  
351 and AYP are the average functions.

352

### 353 **3.3 Code generation**

354 Given a computation graph, we design a lightweight engine to generate the  
355 corresponding source code (Fig. 8). Each operator node in the computation graph is  
356 called a kernel. The sequence of all kernels in a graph is usually fused into a large kernel  
357 function. Therefore, the underlying engine schedules and executes the fused kernel once  
358 and obtains the final result directly without any auxiliary or temporary variables.  
359 Simultaneously, the scheduling overhead of the computation graph and the startup  
360 overhead of the basic kernels can be reduced.

361

362 Most of the scientific computational applications are limited by the memory bandwidth  
363 and cannot fully exploit the computing power of a processor. Fortunately, kernel fusion  
364 is an effective optimization method to improve memory locality. When two kernels  
365 need to process some data, their fusion holds shared data in the memory. Prior to the  
366 kernel fusion, the computation graph is analysed to find the operator nodes that can be  
367 fused, and the analysis results are stored in several subgraphs. Users can access to any  
368 individual subgraph by assigning the subgraph to an intermediate variable for  
369 diagnostic purposes. After being given a series of subgraphs, the underlying engine  
370 dynamically generates the corresponding kernel function in C++ using just-in-time (JIT)  
371 compilation techniques (Suganuma and Yasue, 2005). The JIT compiler used in  
372 OpenArray can fuse numbers of operators into a large compiled kernel. The benefit of  
373 fusing operators is to alleviate memory bandwidth limitations and improve performance  
374 compared with executing operators one-by-one. In order to generate a kernel function  
375 based on a subgraph, we first add the function header and variable definitions according  
376 to the name and type in the *Array* structure. And then we add the loop head through the  
377 dimension information. Finally, we perform a depth-first walk on the expression tree to  
378 convert data, operators, and assignment nodes into a complete expression including

379 load variables, arithmetic operation, and equal symbol with C++ language.

380

381 Notably, the time to compile a single kernel function is short, but practical applications  
382 usually need to be run for thousands of time steps, and the overhead of generating and  
383 compiling the kernel functions for the computation graph is extremely high. Therefore,  
384 we generate a fusion kernel function only once for each subgraph, and put it into a  
385 function pool. Later, when facing the same computation subgraph, we fetch the  
386 corresponding fusion kernel function directly from the pool.

387

388 Since the arrays in OpenArray are distributed among different processing units, and the  
389 operator needs to use the data in the neighbouring points, in order to ensure the  
390 correctness, it is necessary to check the data consistency before fusion. The use of  
391 different data splitting methods for distributed arrays can greatly affect computing  
392 performance. The current data splitting method in OpenArray is the widely used block-  
393 based strategy. Solving PDEs on structured grids often divides the simulated domain  
394 into blocks that are distributed to different processing units. However, the differential  
395 and average operators always require their neighbouring points to perform array  
396 computations. Clearly, ocean modellers have to frequently call corresponding functions  
397 to carefully control the communication of the local boundary region.

398

399 Therefore, we implemented a general boundary management module to implicitly  
400 maintain and update the local boundary information so that the modellers no longer  
401 need to address the message communication. The boundary management module uses  
402 asynchronous communication to update and maintain the data of the boundary region,  
403 which is useful for simultaneous computing and communication. These procedures of  
404 asynchronous communication are implicitly invoked when calling the basic kernel or  
405 the fused kernel to ensure that the parallel details are completely transparent to the  
406 modellers. For the global boundary conditions of the limited physical domains, the  
407 values at the physical border are always set to zero within the operators and operator

408 expressions. In realistic cases, the global boundary conditions are set by a series of  
409 functions (e.g., radiation, wall) provided by OpenArray.

410

### 411 **3.4 Portable program for different backend platforms**

412 With the help of dynamic code generation and JIT compilation technology, OpenArray  
413 can be migrated to different backend platforms. Several basic libraries, including Boost  
414 C++ libraries and Armadillo library, are required. The JIT compilation module is based  
415 on Low-Level-Virtual-Machine (LLVM), thus theoretically the module can only be  
416 ported to platforms supporting LLVM. If LLVM is not supported, as on the Sunway  
417 platform, one can generate the fusion kernels in advance by running the ocean model  
418 on an X86 platform. If the target platform is CPUs with acceleration cards, such as GPU  
419 clusters, it is necessary to add control statements in the CPU code, including data  
420 transmission, calculation, synchronous and asynchronous statements. In addition, the  
421 accelerating solution should involve the selection of the best parameters, for example  
422 “blockDim” and “gridDim” on GPU platforms. In short, the code generation module of  
423 OpenArray also needs to be refactored to be able to generate codes for different backend  
424 platforms. The application based on OpenArray can then be migrated seamlessly to the  
425 target platform. Currently, we have designed the corresponding source code generation  
426 module for Intel CPU and Sunway processors in OpenArray.

427

428 According to the TOP500 list released in November 2018, the Sunway TaihuLight is  
429 ranked third in the world, with a LINPACK benchmark rating of 93 Petaflops provided  
430 by Sunway many-core processors (or Sunway CPUs). As shown in Fig. 9, every  
431 Sunway CPU includes 260 processing elements (or cores) that are divided into 4 core-  
432 groups. Each core-group consists of 64 computing processing elements (CPEs) and a  
433 management processing element (MPE) (Qiao et al., 2017). CPEs handle large-scale  
434 computing tasks and MPE is responsible for the task scheduling and communication.  
435 The relationship between MPE and CPE is like that between CPU and many-core  
436 accelerator, except for they are fused into a single Sunway processor sharing a unified

437 memory space. To make the most of the computing resources of the Sunway TaihuLight,  
438 we generate kernel functions for the MPE, which is responsible for the thread control,  
439 and CPE, which performs the computations. The kernel functions are fully optimized  
440 with several code optimization techniques (Pugh, 1991) such as loop tiling, loop  
441 aligning, single-instruction multiple-data (SIMD) vectorization, and function inline. In  
442 addition, due to the high memory access latency of CPEs, we accelerate data access by  
443 providing instructions for direct memory access in the kernel to transfer data between  
444 the main memory and local memory (Fu et al., 2017).

445

#### 446 **4. Implementation of GOMO**

447 In this section, we introduce how to implement a numerical ocean model using  
448 OpenArray. The most important step is to derive the primitive discrete governing  
449 equations in operator expression form, then the following work is completed by  
450 OpenArray.

451

452 The fundamental equations of GOMO are derived from POM. GOMO features a  
453 bottom-following, free-surface, staggered Arakawa C grid. To effectively evolve the  
454 rapid surface fluctuations, GOMO uses the mode-splitting algorithm inherited from  
455 POM to address the fast propagating surface gravity waves and slow propagating  
456 internal waves in barotropic (external) and baroclinic (internal) modes, respectively.  
457 The details of the continuous governing equations, the corresponding operator  
458 expression form and the descriptions of all the variables used in GOMO are listed in  
459 the Appendix A, Appendix B, and Appendix C, respectively.

460

461 Figure 10 shows the basic flow diagram of GOMO. At the beginning, we initialize  
462 OpenArray to make all operators suitable for GOMO. After loading the initial values  
463 and the model parameters, the distance information is input into the differential  
464 operators through grid binding. In the external mode, the main consumption is  
465 computing the 2D sea surface elevation  $\eta$  and column-averaged velocity ( $Ua, Va$ ). In

466 the internal mode, 3D array computations predominate in order to calculate baroclinic  
467 motions ( $U, V, W$ ), tracers ( $T, S, \rho$ ), and turbulence closure scheme ( $q^2, q^2l$ ) (Mellor and  
468 Yamada, 1982), where ( $U, V, W$ ) are the velocity fields in the  $x, y$  and  $\sigma$  directions, ( $T,$   
469  $S, \rho$ ) are the potential temperature, the salinity and the density. ( $q^2/2, q^2l/2$ ) are the  
470 turbulence kinetic energy and production of turbulence kinetic energy with turbulence  
471 length scale.

472

473 When the user dives into the GOMO code, the main time stepping loop in GOMO  
474 appears to run on a single processor. However, as described above, implicit parallelism  
475 is the most prominent feature of the program using OpenArray. The operators in  
476 OpenArray, not only the difference and average operators, but also the “+”, “-”, “\*”, “/”  
477 and “=” operators in the Fortran code, are all overloaded for the special data structure  
478 “Array”. The seemingly serial Fortran code is implicitly converted to parallel C++ code  
479 by OpenArray, and the parallelization is hidden from the modellers.

480

481 Because the complicated parallel optimization and tuning processes are decoupled from  
482 the ocean modelling, we completely implemented GOMO based on OpenArray in only  
483 4 weeks, whereas implementation may take several months or even longer when using  
484 the MPI or CUDA library.

485

486 In comparison with the existing POM and its multiple variations, to name a few, Stony  
487 Brook Parallel Ocean Model (sbPOM), mpiPOM and POMgpu, GOMO has less code  
488 but is more powerful in terms of compatibility. As shown in Table 3, the serial version  
489 of POM (POM2k) contains 3521 lines of code. sbPOM and mpiPOM are parallelized  
490 using MPI, while POMgpu is based on MPI and CUDA-C. The codes of sbPOM,  
491 mpiPOM and POMgpu are extended to 4801, 9680 and 30443 lines. In contrast, the  
492 code of GOMO is decreased to 1860 lines. Moreover, GOMO completes the same  
493 function as the other approaches while using the least amount of code (Table 4), since  
494 the complexity has been transferred to OpenArray, which includes about 11,800 lines

495 of codes.

496

497 In addition, poor portability considerably restricts the use of advanced hardware in  
498 oceanography. With the advantages of OpenArray, GOMO is adaptable to different  
499 hardware architectures, such as the Sunway processor. The modellers do not need to  
500 modify any code when changing platforms, eliminating the heavy burden of  
501 transmitting code. As computing platforms become increasingly diverse and complex,  
502 GOMO becomes more powerful and attractive than the machine-dependent models.

503

## 504 **5. Results**

505 In this section, we first evaluate the basic performance of OpenArray using benchmark  
506 tests on a single CPU platform. After checking the correctness of GOMO through an  
507 ideal seamount test case, we use GOMO to further test the scalability and efficiency of  
508 OpenArray.

509

### 510 **5.1 Benchmark testing**

511 We choose two typical PDEs and their implementations from Rodinia v3.1, which is a  
512 benchmark suite for heterogeneous computing (Che et al., 2009), as the original version.  
513 For comparison, we re-implement these two PDEs using OpenArray. In addition, we  
514 added two other test cases. As shown in Table 5, the 2D continuity equation is used to  
515 solve sea surface height, and its continuous form is shown in Eq. (1). The 2D heat  
516 diffusion equation is a parabolic PDE that describes the distribution of heat over time  
517 in a given region. Hotspot is a thermal simulation used for estimating processor  
518 temperature on structured grids (Che et al., 2009; Huang et al., 2006). We tested one  
519 2D case (Hotspot2D) and one 3D case (Hotspot3D) of this program. The average  
520 runtime for 100 iterations is taken as the performance metric. All tests are executed on  
521 a single workstation with an Intel Xeon E5-2650 CPU. The experimental results show  
522 that the performance of OpenArray versions is comparable to the original versions.

523



## 524 **5.2 Validation tests of GOMO**

525 The seamount problem proposed by Beckman and Haidvogel is a widely used ideal test  
526 case for regional ocean models (Beckmann and Haidvogel, 1993). It is a stratified  
527 Taylor column problem, which simulates the flow over an isolated seamount with a  
528 constant salinity and a reference vertical temperature stratification. An eastward  
529 horizontal current of 0.1 m/s is added at model initialization. The southern and northern  
530 boundaries are closed. If the Rossby number is small, an obvious anticyclonic  
531 circulation is trapped by the mount in the deep water.

532

533 Using the seamount test case, we compare GOMO and sbPOM results. The  
534 configurations of both models are exactly the same. Figure 11 shows that GOMO and  
535 sbPOM both capture the anticyclonic circulation at 3500 metres depth. The shaded plot  
536 shows the surface elevation, and the array plot shows the current at 3500 metres. Figure  
537 11(a), 11(b), and 11(c) are the results of GOMO, sbPOM, and the difference (GOMO-  
538 sbPOM), respectively. The differences in the surface elevation and deep currents  
539 between the two models are negligible (Fig. 11(c)).

540

## 541 **5.3 The weak and strong scalability of GOMO**

542 The seamount test case is used to compare the performance of sbPOM and GOMO in  
543 a parallel environment. We use the X86 cluster at National Supercomputing Center in  
544 Wuxi of China, which provides 5000 Intel Xeon E5-2650 v2 CPUs for our account at  
545 most. Figure 12(a) shows the result of a strong scaling evaluation, in which the model  
546 size is fixed at  $2048 \times 2048 \times 50$ . The dashed line indicates the ideal speedup. For the  
547 largest parallelisms with 4096 processes, GOMO and sbPOM achieve 91% and 92%  
548 parallel efficiency, respectively. Figure 12(b) shows the weak scalability of sbPOM and  
549 GOMO. In the weak scaling test, the model size for each process is fixed at  $128 \times 128 \times 50$ ,  
550 and the number of processes is gradually increased from 16 to 4096. Taking the  
551 performance of 16 processes as a baseline, we determine that the parallel efficiencies  
552 of GOMO and sbPOM using 4096 processes are 99.0% and 99.2%, respectively.

553

#### 554 **5.4 Testing on the Sunway platform**

555 We also test the scalability of GOMO on the Sunway platform. Supposing that the  
556 baseline is the runtime of GOMO at 10000 Sunway cores with a grid size of  
557  $4096 \times 4096 \times 50$ , the parallel efficiency of GOMO can still reach 85% at 150000 cores,  
558 as shown in Fig. 13. However, we notice that the scalability declines sharply when the  
559 number of cores exceeds 150000. There are two reasons leading to this decline. First,  
560 the block size assigned to each core decreases as the number of cores increases, causing  
561 more communication during boundary region updating. Second, some processes cannot  
562 be accelerated even though more computing resources are available; for example, the  
563 time spent on creating the computation graph, generating the fusion kernels, and  
564 compiling the JIT cannot be reduced. Even though the fusion-kernel codes are  
565 generated and compiled only once at the beginning of a job, it consumes about 2  
566 minutes. In a sense, OpenArray performs better when processing large-scale data, and  
567 GOMO is more suitable for high-resolution scenarios. In the future, we will further  
568 optimize the communication and graph-creating modules to improve the efficiency for  
569 large-scale cores.

570

#### 571 **6. Discussion**

572 As we mentioned in Section 1, the advantages of OpenArray are easy-to-use, high  
573 efficiency and portability. Using OpenArray, the modellers without any parallel  
574 computing skill and experience can write simple operator expressions in Fortran to  
575 implement complex ocean models. The ocean models can be run on any CPU and  
576 Sunway platforms which have deployed the OpenArray library. We call this effect  
577 “write once, run everywhere”. Other similar libraries (e.g., ATMOL, ICON DSL, and  
578 STELLA, COARRAY) require the users to manually control the boundary  
579 communication and task scheduling to some extent. In contrast, OpenArray implements  
580 completely implicit parallelism with user-friendly interfaces and programming  
581 languages.

582

583 However, there are still several problems to be solved in the development of OpenArray.  
584 The first issue is computational efficiency. Once a variable is in one of the processor  
585 registers or in the highest speed cache, it should be used as much as possible before  
586 being replaced. In fact, we should never to move variables more than once each  
587 timestep. The memory consumption brought by overloading techniques is usually high  
588 due to the unnecessary variable moving and unavoidable cache missing. The current  
589 efficiency and scalability of GOMO are close to sbPOM, since we have adopted a series  
590 of optimization methods, such as memory pool, graph computing, JIT compilation, and  
591 vectorization, to alleviate the requirement of memory bandwidth. However, we have to  
592 admit that we cannot fully solve the memory bandwidth limited problem at present. We  
593 think that time skewing is a cache oblivious algorithm for stencil computations (Frigo  
594 and Strumpen, 2005), since it can exploit temporal locality optimally throughout the  
595 entire memory hierarchy. In addition, the polyhedral model may be another potential  
596 approach, which uses an abstract mathematical representation based on integer  
597 polyhedral, to analyze and optimize the memory access pattern of a program.

598

599 The second issue is that the current OpenArray version cannot support customized  
600 operators. When modellers try out another higher-order advection or any other  
601 numerical scheme, the twelve basic operators provided by OpenArray are not abundant.  
602 We consider using a template mechanism to support the customized operators. The  
603 rules of operations are defined in a template file, where the calculation form of each  
604 customized operator is described by a regular expression. If users want to add a  
605 customized operator, they only need to append a regular expression into the template  
606 file.

607

608 OpenArray and GOMO will continue to be developed, and the following three key  
609 improvements are planned for the following years.

610

611 First, we are developing the GPU version of OpenArray. During the development, the  
612 principle is to keep hot data staying in GPU memory or directly swapping between  
613 GPUs and avoid returning data to the main CPU memory. NVLink provides high  
614 bandwidth and outstanding scalability for GPU-to-CPU or GPU-to-GPU  
615 communication, and addresses the interconnect issue for multi-GPU and multi-  
616 GPU/CPU systems.

617

618 Second, the data Input/Output is becoming a bottleneck of earth system models as the  
619 resolution increases rapidly. At present we encapsulate the PnetCDF library to provide  
620 simple I/O interfaces, such as load operation and store operation. A climate fast  
621 input/output (CFIO) library (Huang et al., 2014) will be implemented into OpenArray  
622 in the next few years. The performance of CFIO is approximately 220% faster than  
623 PnetCDF because of the overlapping of I/O and computing. CFIO will be merged into  
624 the future version of OpenArray and the performance is expected to be further improved.

625

626 Finally, as most of the ocean models, GOMO also faces the load imbalance issue. We  
627 are adding the more effective load balance schemes, including space-filling curve  
628 (Dennis, 2007) and curvilinear orthogonal grids, into OpenArray in order to reduce the  
629 computational cost on land points.

630

631 OpenArray is a product of collaboration between oceanographers and computer  
632 scientists. It plays an important role to simplify the porting work on the Sunway  
633 TaihuLight supercomputer. We believe that OpenArray and GOMO will continue to be  
634 maintained and upgraded. We aim to promote it to the model community as a  
635 development tool for future numerical models.

636

## 637 **7. Conclusion**

638 In this paper, we design a simple computing library (OpenArray) to decouple ocean  
639 modelling and parallel computing. OpenArray provides twelve basic operators that are

640 abstracted from PDEs and extended to ocean model governing equations. These  
641 operators feature user-friendly interfaces and an implicit parallelization ability.  
642 Furthermore, some state-of-art optimization mechanisms, including computation  
643 graphing, kernel fusion, dynamic source code generation and JIT compiling, are applied  
644 to boost the performance. The experimental results prove that the performance of a  
645 program using OpenArray is comparable to that of well-designed programs using  
646 Fortran. Based on OpenArray, we implement a numerical ocean model (GOMO) with  
647 high productivity, enhanced readability and excellent scalable performance. Moreover,  
648 GOMO shows high scalability on both CPU and the Sunway platform. Although more  
649 realistic tests are needed, OpenArray may signal the beginning of a new frontier in  
650 future ocean modelling through ingesting basic operators and cutting-edge computing  
651 techniques.

652

653 *Code availability.* The source codes of OpenArray v1.0 is available at  
654 <https://github.com/hxmhuang/OpenArray>, and the user manual of OpenArray can be  
655 accessed at <https://github.com/hxmhuang/OpenArray/tree/master/doc>. GOMO is  
656 available at <https://github.com/hxmhuang/GOMO>.

657

## 658 **Appendix A: Continuous governing equations**

659 The equations governing the baroclinic (internal) mode in GOMO are the 3-  
660 dimensional hydrostatic primitive equations.

$$661 \quad \frac{\partial \eta}{\partial t} + \frac{\partial UD}{\partial x} + \frac{\partial VD}{\partial y} + \frac{\partial W}{\partial \sigma} = 0, \quad (\text{A1})$$

$$662 \quad \frac{\partial UD}{\partial t} + \frac{\partial U^2 D}{\partial x} + \frac{\partial UV D}{\partial y} + \frac{\partial UW}{\partial \sigma} - fVD + gD \frac{\partial \eta}{\partial x} = \frac{\partial}{\partial \sigma} \left( \frac{K_M}{D} \frac{\partial U}{\partial \sigma} \right) +$$

$$663 \quad \frac{gD^2}{\rho_0} \frac{\partial}{\partial x} \int_{\sigma}^0 \rho d\sigma' - \frac{gD}{\rho_0} \frac{\partial D}{\partial x} \int_{\sigma}^0 \sigma' \frac{\partial \rho}{\partial \sigma'} d\sigma' + F_u, \quad (\text{A2})$$

$$664 \quad \frac{\partial VD}{\partial t} + \frac{\partial UV D}{\partial x} + \frac{\partial V^2 D}{\partial y} + \frac{\partial VW}{\partial \sigma} + fUD + gD \frac{\partial \eta}{\partial y} = \frac{\partial}{\partial \sigma} \left( \frac{K_M}{D} \frac{\partial V}{\partial \sigma} \right) +$$

$$665 \quad \frac{gD^2}{\rho_0} \frac{\partial}{\partial y} \int_{\sigma}^0 \rho d\sigma' - \frac{gD}{\rho_0} \frac{\partial D}{\partial y} \int_{\sigma}^0 \sigma' \frac{\partial \rho}{\partial \sigma'} d\sigma' + F_v, \quad (\text{A3})$$

$$666 \quad \frac{\partial TD}{\partial t} + \frac{\partial TUD}{\partial x} + \frac{\partial TVD}{\partial y} + \frac{\partial TW}{\partial \sigma} = \frac{\partial}{\partial \sigma} \left( K_H \frac{\partial T}{\partial \sigma} \right) + F_T + \frac{\partial R}{\partial \sigma}, \quad (\text{A4})$$

$$667 \quad \frac{\partial SD}{\partial t} + \frac{\partial SUD}{\partial x} + \frac{\partial SVD}{\partial y} + \frac{\partial SW}{\partial \sigma} = \frac{\partial}{\partial \sigma} \left( K_H \frac{\partial S}{\partial \sigma} \right) + F_S, \quad (\text{A5})$$

$$668 \quad \rho = \rho(T, S, p), \quad (\text{A6})$$

$$669 \quad \frac{\partial q^2 D}{\partial t} + \frac{\partial U q^2 D}{\partial x} + \frac{\partial V q^2 D}{\partial y} + \frac{\partial W q^2}{\partial \sigma} = \frac{\partial}{\partial \sigma} \left( \frac{K_q}{D} \frac{\partial q^2}{\partial \sigma} \right) + \frac{2K_M}{D} \left[ \left( \frac{\partial U}{\partial \sigma} \right)^2 + \left( \frac{\partial V}{\partial \sigma} \right)^2 \right] +$$

$$670 \quad \frac{2g}{\rho_0} K_H \frac{\partial \rho}{\partial \sigma} - \frac{2Dq^3}{B_1 l} + F_{q^2}, \quad (\text{A7})$$

$$671 \quad \frac{\partial q^2 l D}{\partial t} + \frac{\partial U q^2 l D}{\partial x} + \frac{\partial V q^2 l D}{\partial y} + \frac{\partial W q^2 l}{\partial \sigma} = \frac{\partial}{\partial \sigma} \left( \frac{K_q}{D} \frac{\partial q^2 l}{\partial \sigma} \right) + E_1 l \left\{ \frac{K_M}{D} \left[ \left( \frac{\partial U}{\partial \sigma} \right)^2 + \right. \right.$$

$$672 \quad \left. \left. \left( \frac{\partial V}{\partial \sigma} \right)^2 \right] + \frac{gE_3}{\rho_0} K_H \frac{\partial \rho}{\partial \sigma} \right\} \tilde{W} - \frac{Dq^3}{B_1} + F_{q^2 l}, \quad (\text{A8})$$

673

674 where  $F_u$ ,  $F_v$ ,  $F_{q^2}$ , and  $F_{q^2 l}$  are horizontal kinematic viscosity terms of  $u$ ,  $v$ ,  $q^2$ , and  
675  $q^2 l$ , respectively.  $F_T$  and  $F_S$  are horizontal diffusion terms of  $T$  and  $S$  respectively.  $\tilde{W}$   
676 is the wall proximity function.

$$677 \quad F_u = \frac{\partial}{\partial x} (2A_M D \frac{\partial U}{\partial x}) + \frac{\partial}{\partial y} \left[ A_M D \left( \frac{\partial U}{\partial y} + \frac{\partial V}{\partial x} \right) \right], \quad (\text{A9})$$

$$678 \quad F_v = \frac{\partial}{\partial y} (2A_M D \frac{\partial V}{\partial y}) + \frac{\partial}{\partial x} \left[ A_M D \left( \frac{\partial U}{\partial y} + \frac{\partial V}{\partial x} \right) \right], \quad (\text{A10})$$

$$679 \quad F_T = \frac{\partial}{\partial x} (A_H H \frac{\partial T}{\partial x}) + \frac{\partial}{\partial y} (A_H H \frac{\partial T}{\partial y}), \quad (\text{A11})$$

$$680 \quad F_S = \frac{\partial}{\partial x} (A_H H \frac{\partial S}{\partial x}) + \frac{\partial}{\partial y} (A_H H \frac{\partial S}{\partial y}), \quad (\text{A12})$$

$$681 \quad F_{q^2} = \frac{\partial}{\partial x} (A_M H \frac{\partial q^2}{\partial x}) + \frac{\partial}{\partial y} (A_M H \frac{\partial q^2}{\partial y}), \quad (\text{A13})$$

$$682 \quad F_{q^2 l} = \frac{\partial}{\partial x} (A_M H \frac{\partial q^2 l}{\partial x}) + \frac{\partial}{\partial y} (A_M H \frac{\partial q^2 l}{\partial y}), \quad (\text{A14})$$

$$683 \quad \tilde{W} = 1 + \frac{E_2 l}{\kappa} \left( \frac{1}{\eta - z} + \frac{1}{H - z} \right). \quad (\text{A15})$$

684 The equations governing the barotropic (external) mode in GOMO are obtained by  
685 vertically integrating the baroclinic equations.

$$686 \quad \frac{\partial \eta}{\partial t} + \frac{\partial U_{AD}}{\partial x} + \frac{\partial V_{AD}}{\partial y} = 0, \quad (\text{A16})$$

$$687 \quad \frac{\partial U_{AD}}{\partial t} + \frac{\partial (U_A)^2 D}{\partial x} + \frac{\partial U_A V_{AD}}{\partial y} - f V_{AD} + g D \frac{\partial \eta}{\partial x} = \tilde{F}_{u_a} - wu(0) +$$

$$688 \quad wu(-1) - \frac{gD}{\rho_0} \int_{-1}^0 \int_{\sigma}^0 \left[ D \frac{\partial \rho}{\partial x} - \frac{\partial D}{\partial x} \sigma' \frac{\partial \rho}{\partial \sigma} \right] d\sigma' d\sigma + G_{u_a}, \quad (\text{A17})$$

$$689 \quad \frac{\partial V_{AD}}{\partial t} + \frac{\partial U_A V_{AD}}{\partial y} + \frac{\partial (V_A)^2 D}{\partial y} + f U_{AD} + g D \frac{\partial \eta}{\partial y} = \tilde{F}_{v_a} - wv(0) +$$

$$690 \quad wv(-1) - \frac{gD}{\rho_0} \int_{-1}^0 \int_{\sigma}^0 \left[ D \frac{\partial \rho}{\partial y} - \frac{\partial D}{\partial y} \sigma' \frac{\partial \rho}{\partial \sigma} \right] d\sigma' d\sigma + G_{v_a}, \quad (\text{A18})$$

691

692 where  $\tilde{F}_{u_a}$  and  $\tilde{F}_{v_a}$  are the horizontal kinematic viscosity terms of  $U_A$  and  $V_A$   
693 respectively.  $G_{u_a}$  and  $G_{v_a}$  are the dispersion terms of  $U_A$  and  $V_A$  respectively. The  
694 subscript 'A' denotes vertical integration.

695

$$696 \quad \tilde{F}_{u_a} = \frac{\partial}{\partial x} \left[ 2H(AA_M) \frac{\partial U_A}{\partial x} \right] + \frac{\partial}{\partial y} \left[ H(AA_M) \left( \frac{\partial U_A}{\partial y} + \frac{\partial V_A}{\partial x} \right) \right], \quad (\text{A19})$$

$$697 \quad \tilde{F}_{v_a} = \frac{\partial}{\partial y} \left[ 2H(AA_M) \frac{\partial V_A}{\partial y} \right] + \frac{\partial}{\partial x} \left[ H(AA_M) \left( \frac{\partial U_A}{\partial y} + \frac{\partial V_A}{\partial x} \right) \right], \quad (\text{A20})$$

$$698 \quad G_{u_a} = \frac{\partial^2 (U_A)^2 D}{\partial x^2} + \frac{\partial^2 U_A V_A D}{\partial x \partial y} - \tilde{F}_{u_a} - \frac{\partial^2 (U^2)_{AD}}{\partial x^2} - \frac{\partial^2 (UV)_{AD}}{\partial y^2} + (F_u)_A, \quad (\text{A21})$$

$$699 \quad G_{v_a} = \frac{\partial^2 U_A V_A D}{\partial x \partial y} + \frac{\partial^2 (V_A)^2 D}{\partial y^2} - \tilde{F}_{v_a} - \frac{\partial^2 (UV)_{AD}}{\partial x^2} - \frac{\partial^2 (V^2)_{AD}}{\partial y^2} + (F_v)_A, \quad (\text{A22})$$

$$700 \quad U_A = \int_{-1}^0 U d\sigma, \quad (\text{A23})$$

$$701 \quad V_A = \int_{-1}^0 V d\sigma, \quad (\text{A24})$$

$$702 \quad (U^2)_A = \int_{-1}^0 U^2 d\sigma, \quad (\text{A25})$$

$$703 \quad (UV)_A = \int_{-1}^0 UV d\sigma, \quad (\text{A26})$$

$$704 \quad (V^2)_A = \int_{-1}^0 V^2 d\sigma, \quad (\text{A27})$$

$$705 \quad (F_u)_A = \int_{-1}^0 F_u d\sigma, \quad (\text{A28})$$

$$706 \quad (F_v)_A = \int_{-1}^0 F_v d\sigma, \quad (\text{A29})$$

$$707 \quad AA_M = \int_{-1}^0 (A_M) d\sigma. \quad (\text{A30})$$

708

## 709 **Appendix B: Discrete governing equations**

710 The discrete governing equations of baroclinic (internal) mode expressed by operators  
711 are shown as below:

$$712 \quad \frac{\eta^{t+1} - \eta^{t-1}}{2dti} + \delta_f^x (\bar{D}_b^x U) + \delta_f^y (\bar{D}_b^y V) + \delta_f^z (W) = 0, \quad (\text{B1})$$

$$\begin{aligned}
713 \quad & \frac{(\overline{D_b^x U})^{t+1} - (\overline{D_b^x U})^{t-1}}{2dti} + \delta_b^x \left[ \overline{(\overline{D_b^x U})_f \overline{U}_f^x} \right] + \delta_f^y \left[ \overline{(\overline{D_b^y V})_b \overline{U}_b^y} \right] + \\
714 \quad & \delta_f^z \left( \overline{W_b^x \overline{U}_b^z} - \overline{(\tilde{f} \overline{V}_f^y D)_b} - \overline{(\tilde{f} \overline{V}_f^y D)_b} + g \overline{D_b^x} \delta_b^x(\eta) \right) = \delta_b^z \left[ \frac{K_{M_b}^x}{(\overline{D_b^x})^{t+1}} \delta_f^z(U^{t+1}) \right] + \\
715 \quad & \frac{g(\overline{D_b^x})^2}{\rho_0} \int_{\sigma}^0 \left[ \delta_b^x(\overline{\rho}_b^z) - \frac{\sigma}{\overline{D_b^x}} \delta_b^x(D) \delta_b^z(\overline{\rho}_b^x) \right] d\sigma' + F_u, \tag{B2}
\end{aligned}$$

$$\begin{aligned}
716 \quad & \frac{(\overline{D_b^y V})^{t+1} - (\overline{D_b^y V})^{t-1}}{2dti} + \delta_f^x \left[ \overline{(\overline{D_b^x U})_b \overline{V}_b^x} \right] + \delta_b^y \left[ \overline{(\overline{D_b^y V})_f \overline{V}_f^y} \right] + \\
717 \quad & \delta_f^z \left( \overline{W_b^y \overline{V}_b^z} + \overline{(\tilde{f} \overline{U}_f^x D)_b} + \overline{(\tilde{f} \overline{U}_f^x D)_b} + g \overline{D_b^y} \delta_b^y(\eta) \right) = \delta_b^z \left[ \frac{K_{M_b}^y}{(\overline{D_b^y})^{t+1}} \delta_f^z(V^{t+1}) \right] + \\
718 \quad & \frac{g(\overline{D_b^y})^2}{\rho_0} \int_{\sigma}^0 \left[ \delta_b^y(\overline{\rho}_b^z) - \frac{\sigma}{\overline{D_b^y}} \delta_b^y(D) \delta_b^z(\overline{\rho}_b^y) \right] d\sigma' + F_v, \tag{B3}
\end{aligned}$$

$$\begin{aligned}
719 \quad & \frac{(TD)^{t+1} - (TD)^{t-1}}{2dti} + \delta_f^x(\overline{T}_b^x U \overline{D}_b^x) + \delta_f^y(\overline{T}_b^y V \overline{D}_b^y) + \delta_f^z(\overline{T}_b^z W) = \\
720 \quad & \delta_b^z \left[ \frac{K_H}{D^{t+1}} \delta_f^z(T^{t+1}) \right] + F_T + \delta_f^z R, \tag{B4}
\end{aligned}$$

$$\begin{aligned}
721 \quad & \frac{(SD)^{t+1} - (SD)^{t-1}}{2dti} + \delta_f^x(\overline{S}_b^x U \overline{D}_b^x) + \delta_f^y(\overline{S}_b^y V \overline{D}_b^y) + \delta_f^z(\overline{S}_b^z W) = \\
722 \quad & \delta_b^z \left[ \frac{K_H}{D^{t+1}} \delta_f^z(S^{t+1}) \right] + F_S, \tag{B5}
\end{aligned}$$

$$723 \quad \rho = \rho(T, S, p), \tag{B6}$$

$$\begin{aligned}
724 \quad & \frac{(q^2 D)^{t+1} - (q^2 D)^{t-1}}{2dti} + \delta_f^x(\overline{U}_b^z q^2 \overline{D}_b^x) + \delta_f^y(\overline{V}_b^z q^2 \overline{D}_b^y) + \\
725 \quad & \delta_f^z(\overline{W} q^2)_b^z = \delta_b^z \left[ \frac{K_{q_f}^z}{D^{t+1}} \delta_f^z(q^2)^{t+1} \right] + \frac{2K_M}{D} \left\{ \left[ \delta_b^z(\overline{U}_f^x) \right]^2 + \left[ \delta_b^z(\overline{V}_f^y) \right]^2 \right\} + \\
726 \quad & \frac{2g}{\rho_0} K_H \delta_b^z(\rho) - \frac{2Dq^3}{B_1 l} + F_{q^2}, \tag{B7}
\end{aligned}$$

$$\begin{aligned}
727 \quad & \frac{(q^2 l D)^{t+1} - (q^2 l D)^{t-1}}{2dti} + \delta_f^x(\overline{U}_b^z q^2 \overline{l}_b^x \overline{D}_b^x) + \delta_f^y(\overline{V}_b^z q^2 \overline{l}_b^y \overline{D}_b^y) + \\
728 \quad & \delta_f^z(\overline{W} q^2 \overline{l})_b^z = \delta_b^z \left[ \frac{K_{q_f}^z}{D^{t+1}} \delta_f^z(q^2 l)^{t+1} \right] + l E_1 \frac{K_M}{D} \left\{ \left[ \delta_b^z(\overline{U}_f^x) \right]^2 + \left[ \delta_b^z(\overline{V}_f^y) \right]^2 \right\} \tilde{W} + \\
729 \quad & \frac{l E_1 E_3 g}{\rho_0} K_H \delta_b^z(\rho) \tilde{W} - \frac{Dq^3}{B_1} + F_{q^2 l}, \tag{B8}
\end{aligned}$$

730

731 where  $F_u$ ,  $F_v$ ,  $F_{q^2}$ , and  $F_{q^2 l}$  are horizontal kinematic viscosity terms of  $u$ ,  $v$ ,  $q^2$ , and

732  $q^2 l$ , respectively.  $F_T$  and  $F_S$  are horizontal diffusion terms of  $T$  and  $S$  respectively.



$$733 \quad F_u = \delta_b^x [2A_M D \delta_f^x (U^{t-1})] + \delta_f^y \left\{ \overline{(\overline{A_{M_b}}^x)_b}^y \overline{(\overline{D_b}^x)_b}^y [\delta_b^x (V)^{t-1} + \delta_b^y (U)^{t-1}] \right\}, \quad (\text{B9})$$

$$734 \quad F_v = \delta_b^y [2A_M D \delta_f^y (V^{t-1})] + \delta_f^x \left\{ \overline{(\overline{A_{M_b}}^x)_b}^y \overline{(\overline{D_b}^x)_b}^y [\delta_b^x (V)^{t-1} + \delta_b^y (U)^{t-1}] \right\}, \quad (\text{B10})$$

$$735 \quad F_T = \delta_f^x [\overline{A_{H_b}^x} \overline{H_b}^x \delta_b^x (T^{t-1})] + \delta_f^y [\overline{A_{H_b}^y} \overline{H_b}^y \delta_b^y (T^{t-1})], \quad (\text{B11})$$

$$736 \quad F_S = \delta_f^x [\overline{A_{H_b}^x} \overline{H_b}^x \delta_b^x (S^{t-1})] + \delta_f^y [\overline{A_{H_b}^y} \overline{H_b}^y \delta_b^y (S^{t-1})], \quad (\text{B12})$$

$$737 \quad F_{q^2} = \delta_f^x \left[ \overline{(\overline{A_{M_b}}^x)_b}^z \overline{H_b}^x \delta_b^x (q^2)^{t-1} \right] + \delta_f^y \left[ \overline{(\overline{A_{M_b}}^y)_b}^z \overline{H_b}^y \delta_b^y (q^2)^{t-1} \right], \quad (\text{B13})$$

$$738 \quad F_{q^2 l} = \delta_f^x \left[ \overline{(\overline{A_{M_b}}^x)_b}^z \overline{H_b}^x \delta_b^x (q^2 l)^{t-1} \right] + \delta_f^y \left[ \overline{(\overline{A_{M_b}}^y)_b}^z \overline{H_b}^y \delta_b^y (q^2 l)^{t-1} \right]. \quad (\text{B14})$$

739

740 The discrete governing equations of barotropic (external) mode expressed by operators  
741 are shown as below:

$$742 \quad \frac{\eta^{t+1} - \eta^{t-1}}{2dte} + \delta_f^x (\overline{D_b}^x U_A) + \delta_f^y (\overline{D_b}^y V_A) = 0, \quad (\text{B15})$$

$$743 \quad \frac{(\overline{D_b}^x U_A)^{t+1} - (\overline{D_b}^x U_A)^{t-1}}{2dte} + \delta_b^x \left[ \overline{(\overline{D_b}^x U_A)_f}^x \overline{(U_A)_f}^x \right] + \delta_f^y \left[ \overline{(\overline{D_b}^y V_A)_b}^x \overline{(U_A)_b}^y \right] -$$

$$744 \quad \left[ \overline{\tilde{f}_A (V_A)_f}^y D \right]_b^x - \left[ \overline{f (V_A)_f}^y D \right]_b^x + g \overline{D_b}^x \delta_b^x (\eta) = \delta_b^x \{ 2(AA_M) D \delta_f^x [(U_A)^{t-1}] \} +$$

$$745 \quad \delta_f^y \left\{ \left[ \overline{(\overline{AA_M})_b}^x \right]_b^y \overline{(\overline{D_b}^x)_b}^y [\delta_b^x (V_A) + \delta_b^y (U_A)]^{t-1} \right\} + \phi_x, \quad (\text{B16})$$

$$746 \quad \frac{(\overline{D_b}^y V_A)^{t+1} - (\overline{D_b}^y V_A)^{t-1}}{2dte} + \delta_f^x \left[ \overline{(\overline{D_b}^x U_A)_b}^y \overline{(V_A)_b}^x \right] + \delta_b^y \left[ \overline{(\overline{D_b}^y V_A)_f}^y \overline{(V_A)_f}^y \right] +$$

$$747 \quad \left[ \overline{\tilde{f}_A (U_A)_f}^x D \right]_b^y + \left[ \overline{f (U_A)_f}^x D \right]_b^y + g \overline{D_b}^y \delta_b^y (\eta) = \delta_b^y \{ 2(AA_M) D \delta_f^y [(V_A)^{t-1}] \} +$$

$$748 \quad \delta_f^x \left\{ \left[ \overline{(\overline{AA_M})_b}^x \right]_b^y \overline{(\overline{D_b}^x)_b}^y [\delta_b^x (V_A) + \delta_b^y (U_A)]^{t-1} \right\} + \phi_y, \quad (\text{B17})$$

749

750 where

$$751 \quad \phi_x = -WU(0) + WU(-1) - \frac{g(\overline{D_b}^x)^2}{\rho_0} \int_{-1}^0 \left\{ \left[ \int_{\sigma}^0 \delta_b^x (\overline{\rho})_b^z d\sigma' \right] d\sigma \right\} +$$

$$752 \quad \frac{g\bar{D}_b^x \delta_b^{xD}}{\rho_0} \int_{-1}^0 \left\{ \left[ \int_{\sigma}^0 \bar{\sigma}_b^z \delta_b^z(\bar{\rho}_b^x) \right] d\sigma \right\} + G_x, \quad (\text{B18})$$

$$753 \quad \phi_y = -WV(0) + WV(-1) - \frac{g(\bar{D}_b^y)^2}{\rho_0} \int_{-1}^0 \left\{ \left[ \int_{\sigma}^0 \delta_b^y(\bar{\rho})_b^z d\sigma' \right] d\sigma \right\} +$$

$$754 \quad \frac{g\bar{D}_b^y \delta_b^{yD}}{\rho_0} \int_{-1}^0 \left\{ \left[ \int_{\sigma}^0 \bar{\sigma}_b^z \delta_b^z(\bar{\rho}_b^y) \right] d\sigma \right\} + G_y. \quad (\text{B19})$$

755

756

757 **Appendix C: Descriptions of symbols**

758 The description of each symbol in the governing equations is list as below:

759 Table C1. Descriptions of symbols

Symbol	Description
$\eta$	Free surface elevation
H	Bottom topography
$u_a, v_a$	Vertical average velocity in x, y direction, respectively
U, V, W	Velocity in x, y, $\sigma$ direction, respectively
D	Fluid column depth
f	The Coriolis parameter
g	The gravitational acceleration
$\rho_0$	Constant density
$\rho$	Situ density
T	Potential temperature
S	Salinity
R	Surface solar radiation incident
$q^2/2$	Turbulence kinetic energy
l	Turbulence length scale
$q^2 l/2$	Production of turbulence kinetic energy and turbulence length scale
dti	Time step of baroclinic mode
dte	Time step of barotropic mode
dx	Grid increment in x direction
dy	Grid increment in y direction
$A_M$	Horizontal kinematic viscosity
$A_H$	Horizontal heat diffusivity
$K_M$	Vertical kinematic viscosity
$K_H$	Vertical mixing coefficient of heat and salinity
$K_q$	Vertical mixing coefficient of turbulence kinetic energy

761 *Author contributions.* Xiaomeng Huang led the project of OpenArray and the writing  
762 of this paper, DW, QW, SZ and Xing Huang designed OpenArray. Xing Huang, DW,  
763 QW, SZ, MW, YG, and QT implemented and tested GOMO. All coauthors contributed  
764 to the writing of this paper.

765

766 *Competing interests.* The authors declare that they have no conflict of interest.

767

768 *Acknowledgements.* Xiaomeng Huang is supported by a grant from the State's Key  
769 Project of Research and Development Plan (2016YFB0201100), ~~and~~ the National  
770 Natural Science Foundation of China (41776010), and Center for High Performance  
771 Computing and System Simulation of Pilot National Laboratory for Marine Science  
772 and Technology (Qingdao). Xing Huang is supported by a grant from the State's Key  
773 Project of Research and Development Plan (2018YFB0505000). Shixun Zhang is  
774 supported by a grant from the State's Key Project of Research and Development Plan  
775 (2017YFC1502200) and Qingdao National Laboratory for Marine Science and  
776 Technology (QNLM2016ORP0108). Zhenya Song is supported by National Natural  
777 Science Foundation of China (U1806205) and AoShan Talents Cultivation Excellent  
778 Scholar Program Supported by Qingdao National Laboratory for Marine Science and  
779 Technology (2017ASTCP-ES04).

780

## 781 **References**

782 Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat,  
783 S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray,  
784 D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y. and  
785 Zheng, X.: TensorFlow: A System for Large-Scale Machine Learning, in 12th  
786 {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI}  
787 16), pp. 265–283, {USENIX} Association, Savannah, GA. [online] Available from:  
788 <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>,  
789 2016.

790 Alexander, K. and Easterbrook, S. M.: The software architecture of climate models: A  
791 graphical comparison of CMIP5 and EMICAR5 configurations, *Geosci. Model Dev.*,  
792 8(4), 1221–1232, doi:10.5194/gmd-8-1221-2015, 2015.

793 Arakawa, A. and Lamb, V. R.: A Potential Enstrophy and Energy Conserving Scheme  
794 for the Shallow Water Equations, *Mon. Weather Rev.*, doi:10.1175/1520-  
795 0493(1981)109<0018:APEAEC>2.0.CO;2, 1981.

796 Bae, H., Mustafa, D., Lee, J. W., Aurangzeb, Lin, H., Dave, C., Eigenmann, R. and  
797 Midkiff, S. P.: The Cetus source-to-source compiler infrastructure: Overview and  
798 evaluation, in *International Journal of Parallel Programming.*, 2013.

799 Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A.,  
800 Bouchard, N., Warde-Farley, D. and Bengio, Y.: Theano: new features and speed  
801 improvements, *CoRR*, abs/1211.5 [online] Available from:  
802 <http://arxiv.org/abs/1211.5590>, 2012.

803 Beckmann, A. and Haidvogel, D. B.: Numerical simulation of flow around a tall  
804 isolated seamount. Part I: problem formulation and model accuracy, *J. Phys.*  
805 *Oceanogr.*, 23(8), 1736–1753, doi:10.1175/1520-  
806 0485(1993)023<1736:NSOFAA>2.0.CO;2, 1993.

807 Bloss, A., Hudak, P. and Young, J.: Code optimizations for lazy evaluation, *Lisp Symb.*  
808 *Comput.*, doi:10.1007/BF01806169, 1988.

809 Blumberg, A. F. and Mellor, G. L.: A description of a three-dimensional coastal ocean  
810 circulation model, , (January 1987), 1–16, doi:10.1029/CO004p0001, 1987.

811 Bonan, G. B. and Doney, S. C.: Climate, ecosystems, and planetary futures: The  
812 challenge to predict life in Earth system models, *Science* (80-. ),  
813 doi:10.1126/science.aam8328, 2018.

814 Bretherton, C., Balaji, V. and Delworth, T. et al: A National Strategy for Advancing  
815 Climate Modeling, National Academies Press., 2012.

816 Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S. H. and Skadron, K.:  
817 Rodinia: A benchmark suite for heterogeneous computing, in *Proceedings of the*  
818 *2009 IEEE International Symposium on Workload Characterization, IISWC 2009.*,

819 2009.

820 Chen, C., Liu, H. and Beardsley, R. C.: An unstructured grid, finite-volume, three-  
821 dimensional, primitive equations ocean model: Application to coastal ocean and  
822 estuaries, *J. Atmos. Ocean. Technol.*, doi:10.1175/1520-  
823 0426(2003)020<0159:AUGFVT>2.0.CO;2, 2003.

824 Collins, M., Minobe, S., Barreiro, M., Bordoni, S., Kaspi, Y., Kuwano-Yoshida, A.,  
825 Keenlyside, N., Manzini, E., O'Reilly, C. H., Sutton, R., Xie, S. P. and Zolina, O.:  
826 Challenges and opportunities for improved understanding of regional climate  
827 dynamics, *Nat. Clim. Chang.*, 8(2), 101–108, doi:10.1038/s41558-017-0059-8, 2018.

828 Corliss, G. and Griewank, A.: Operator Overloading as an Enabling Technology for  
829 Automatic Differentiation, 1994.

830 Deconinck, W., Bauer, P., Diamantakis, M., Hamrud, M., Kühnlein, C., Maciel, P.,  
831 Mengaldo, G., Quintino, T., Raoult, B., Smolarkiewicz, P. K. and Wedi, N. P.: Atlas :  
832 A library for numerical weather prediction and climate modelling, *Comput. Phys.*  
833 *Commun.*, 220, 188–204, doi:10.1016/j.cpc.2017.07.006, 2017.

834 Dennis, J. M.: Inverse space-filling curve partitioning of a global ocean model, *Proc. -*  
835 *21st Int. Parallel Distrib. Process. Symp. IPDPS 2007*; Abstr. CD-ROM, 1–10,  
836 doi:10.1109/IPDPS.2007.370215, 2007.

837 van Engelen, R. a.: ATMOL: A Domain-Specific Language for Atmospheric Modeling,  
838 *J. Comput. Inf. Technol.*, 9(4), 289–303, doi:10.2498/cit.2001.04.02, 2001.

839 Frigo, M. and Strumpen, V.: Cache oblivious stencil computations, , 361,  
840 doi:10.1145/1088149.1088197, 2005.

841 Fu, H., He, C., Chen, B., Yin, Z., Zhang, Z., Zhang, W., Zhang, T., Xue, W., Liu, W.,  
842 Yin, W. and others: 18.9-Pflops nonlinear earthquake simulation on Sunway  
843 TaihuLight: enabling depiction of 18-Hz and 8-meter scenarios, in *Proceedings of*  
844 *the International Conference for High Performance Computing, Networking, Storage*  
845 *and Analysis.*, 2017.

846 Griffies, S. M., Böning, C., Bryan, F. O., Chassignet, E. P., Gerdes, R., Hasumi, H.,  
847 Hirst, A., Treguier, A.-M. and Webb, D.: Developments in ocean climate modelling,

848 Ocean Model., 2(3–4), 123–192, doi:10.1016/S1463-5003(00)00014-7, 2000.

849 Gysi, T., Osuna, C., Fuhrer, O., Bianco, M. and Schulthess, T. C.: STELLA: A Domain-  
850 specific Tool for Structured Grid Methods in Weather and Climate Models, Proc. Int.  
851 Conf. High Perform. Comput. Networking, Storage Anal. - SC '15, 1–12,  
852 doi:10.1145/2807591.2807627, 2015.

853 Huang, W., Ghosh, S., Velusamy, S., Sankaranarayanan, K., Skadron, K. and Stan, M.  
854 R.: HotSpot: A compact thermal modeling methodology for early-stage VLSI design,  
855 IEEE Trans. Very Large Scale Integr. Syst., doi:10.1109/TVLSI.2006.876103, 2006.

856 Huang, X. M., Wang, W. C., Fu, H. H., Yang, G. W., Wang, B. and Zhang, C.: A fast  
857 input/output library for high-resolution climate models, Geosci. Model Dev., 7(1),  
858 93–103, doi:10.5194/gmd-7-93-2014, 2014.

859 Korn, P.: Formulation of an unstructured grid model for global ocean dynamics, J.  
860 Comput. Phys., 339, 525–552, doi:10.1016/j.jcp.2017.03.009, 2017.

861 Lawrence, B. N., Rezný, M., Budich, R., Bauer, P., Behrens, J., Carter, M., Deconinck,  
862 W., Ford, R., Maynard, C., Müllerworth, S., Osuna, C., Porter, A., Serradell, K.,  
863 Valcke, S., Wedi, N. and Wilson, S.: Crossing the chasm: How to develop weather  
864 and climate models for next generation computers?, Geosci. Model Dev.,  
865 doi:10.5194/gmd-11-1799-2018, 2018.

866 Levin, J. G., Iskandarani, M. and Haidvogel, D. B.: A nonconforming spectral element  
867 ocean model, Int. J. Numer. Methods Fluids, 34(6), 495–525, doi:10.1002/1097-  
868 0363(20001130)34:6<495::AID-FLD68>3.0.CO;2-K, 2000.

869 Lidman, J., Quinlan, D. J., Liao, C. and McKee, S. A.: ROSE::FTTransform - A source-  
870 to-source translation framework for exascale fault-tolerance research, Proc. Int. Conf.  
871 Dependable Syst. Networks, (June), doi:10.1109/DSNW.2012.6264672, 2012.

872 Mellor-Crummey, J., Adhianto, L., Scherer III, W. N. and Jin, G.: A New Vision for  
873 Coarray Fortran, in Proceedings of the Third Conference on Partitioned Global  
874 Address Space Programming Models, p. 5:1--5:9, ACM, New York, NY, USA., 2009.

875 Mellor, G. L.: Users guide for a three-dimensional, primitive equation, numerical ocean  
876 model (June 2003 version), Prog. Atmos. Ocean. Sci, Princet. Univ., (October), 53,

877 2003.

878 Mellor, G. L. and Yamada, T.: Development of a turbulence closure model for  
879 geophysical fluid problems, *Rev. Geophys.*, doi:10.1029/RG020i004p00851, 1982.

880 Porkoláb, Z., Mihalicza, J. and Sipos, Á.: Debugging C++ template metaprograms, ,  
881 255, doi:10.1145/1173706.1173746, 2007.

882 Pugh, W.: Uniform Techniques for Loop Optimization, in *Proceedings of the 5th*  
883 *International Conference on Supercomputing*, pp. 341–352, ACM, New York, NY,  
884 USA., 1991.

885 Qiao, F., Zhao, W., Yin, X., Huang, X., Liu, X., Shu, Q., Wang, G., Song, Z., Li, X.,  
886 Liu, H., Yang, G. and Yuan, Y.: A Highly Effective Global Surface Wave Numerical  
887 Simulation with Ultra-High Resolution, in *International Conference for High*  
888 *Performance Computing, Networking, Storage and Analysis, SC.*, 2017.

889 Reynolds, J. C.: *Theories of Programming Languages*, Cambridge University Press,  
890 New York, NY, USA., 1999.

891 Shan, A.: Heterogeneous Processing: A Strategy for Augmenting Moore’s Law, *Linux*  
892 *J.*, 2006(142). Available from: <http://dl.acm.org/citation.cfm?id=1119128.1119135>,  
893 2006.

894 Shchepetkin, A. F. and McWilliams, J. C.: The regional oceanic modeling system  
895 (ROMS): A split-explicit, free-surface, topography-following-coordinate oceanic  
896 model, *Ocean Model.*, doi:10.1016/j.ocemod.2004.08.002, 2005.

897 Suganuma, T. and Yasue, T.: Design and evaluation of dynamic optimizations for a  
898 Java just-in-time compiler, *ACM Trans. ....*, doi:10.1145/1075382.1075386, 2005.

899 Taylor, K. E., Stouffer, R. J. and Meehl, G. A.: An overview of CMIP5 and the  
900 experiment design, *Bull. Am. Meteorol. Soc.*, 93(4), 485–498, doi:10.1175/BAMS-  
901 D-11-00094.1, 2012.

902 Torres, R., Linardakis, L., Kunkel, J. and Ludwig, T.: ICON DSL: A Domain-Specific  
903 Language for climate modeling, *Sc13.Supercomputing.Org* [online] Available from:  
904 <http://sc13.supercomputing.org/sites/default/files/WorkshopsArchive/pdfs/wp127s1>  
905 .pdf, 2013.



906 Walther, A., Griewank, A. and Vogel, O.: ADOL-C: Automatic Differentiation Using  
907 Operator Overloading in C++, PAMM, doi:10.1002/pamm.200310011, 2003.  
908 Xu, S., Huang, X., Oey, L. Y., Xu, F., Fu, H., Zhang, Y. and Yang, G.: POM.GPU-v1.0:  
909 A GPU-based princeton ocean model, Geosci. Model Dev., doi:10.5194/gmd-8-  
910 2815-2015, 2015.  
911

Notations	Discrete Form	Basic Operator
$\overline{var}_f^x$	$[ var(i,j,k) + var(i+1,j,k) ] / 2$	AXF
$\overline{var}_b^x$	$[ var(i,j,k) + var(i-1,j,k) ] / 2$	AXB
$\overline{var}_f^y$	$[ var(i,j,k) + var(i,j+1,k) ] / 2$	AYF
$\overline{var}_b^y$	$[ var(i,j,k) + var(i,j-1,k) ] / 2$	AYB
$\overline{var}_f^z$	$[ var(i,j,k) + var(i,j,k+1) ] / 2$	AZF
$\overline{var}_b^z$	$[ var(i,j,k) + var(i,j,k-1) ] / 2$	AZB
$\delta_f^x(var)$	$[ var(i+1,j,k) - var(i,j,k) ] / dx(i,j)$	DXF
$\delta_b^x(var)$	$[ var(i,j,k) - var(i-1,j,k) ] / dx(i-1,j)$	DXB
$\delta_f^y(var)$	$[ var(i,j+1,k) - var(i,j,k) ] / dy(i,j)$	DYF
$\delta_b^y(var)$	$[ var(i,j,k) - var(i,j-1,k) ] / dy(i,j-1)$	DYB
$\delta_f^z(var)$	$[ var(i,j,k+1) - var(i,j,k) ] / dz(k)$	DZF
$\delta_b^z(var)$	$[ var(i,j,k) - var(i,j,k-1) ] / dz(k-1)$	DZB

916

Table 2 The jumping rules of an operator acting on an *Array*

The initial position of <i>var</i>	The position of $[A/D]X[F/B]$ ( <i>var</i> )	The position of $[A/D]Y[F/B]$ ( <i>var</i> )	The position of $[A/D]Z[F/B]$ ( <i>var</i> )
0	1	2	4
1	0	3	5
2	3	0	6
3	2	1	7
4	5	6	0
5	4	7	1
6	7	4	2
7	6	5	3

917

918

919

Table 3. Comparing GOMO with several variations of the POM

Model	Lines of code	Method	Computing Platforms
POM2k	3521	Serial	CPU
sbPOM	4801	MPI	CPU
mpiPOM	9685	MPI	CPU
POMgpu	30443	MPI + CUDA	GPU
GOMO	1860	OpenArray	CPU, Sunway

920

921

922

Table. 4. Comparison of the amount of code for different functions

Functions	Lines of code		
	POM2k	sbPOM	GOMO
Solve for $\eta$	16	72	<b>1</b>
Solve for Ua	75	183	<b>11</b>
Solve for Va	75	183	<b>11</b>
Solve for W	36	90	<b>3</b>
Solve for $q^2$ and $q^2l$	318	854	<b>162</b>
Solve for T or S	178	234	<b>71</b>
Solve for U	118	230	<b>50</b>
Solve for V	118	230	<b>50</b>

923

924

925

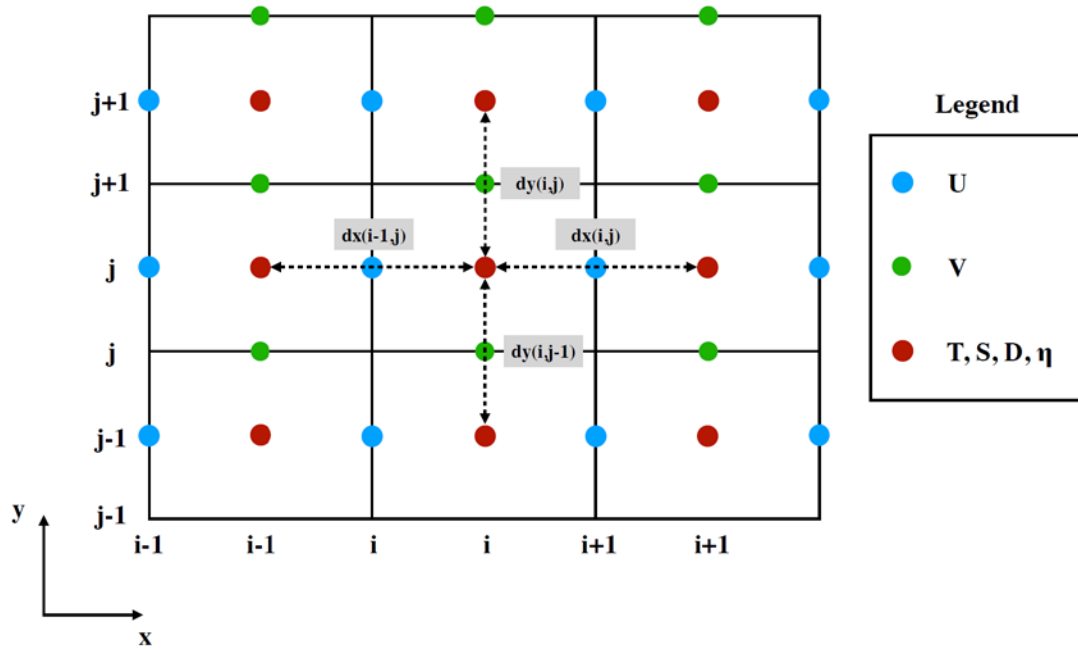
Table 5. Four benchmark tests

Benchmark	Dimensions	Grid Size	OpenArray version (seconds)	Original version(seconds)
Continuity equation	2D	8192×8192	7.22	7.10
Heat diffusion equation	2D	8192×8192	6.20	6.34
Hotspot2D	2D	8192×8192	11.37	11.21
Hotspot3D	3D	512×512×8	0.96	1.01

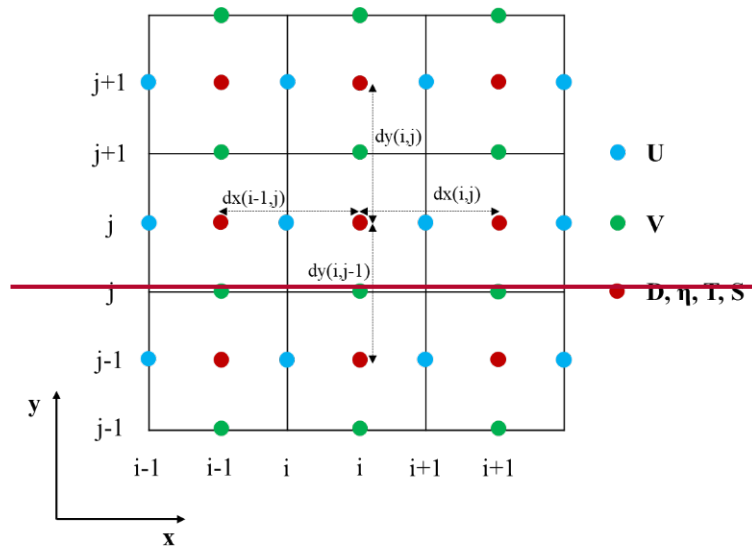
926

927

929



930



931

932

933

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

934

\$ 1) 2D continuous equation

$$\eta_{t+1} = \eta_{t-1} - 2 * dt * ( \delta_x^2 (\bar{D}_b^x * U) + \delta_y^2 (\bar{D}_b^y * V) )$$

\$ 2) The code constructed by operators

$$elf = elb - 2 * dt * (DXF(AXB(D)*U) + DYF(AYB(D)*V))$$

\$ 3) The pseudo-code

```
exchange2d_mpi(u,im,jm)
exchange2d_mpi(v,im,jm)
exchange2d_mpi(D,im,jm)
```

```
do i = 1, im
```

```
  do j = 1, jm
```

```
    elf(i,j) = elb(i,j) - 2 * dt * ( &
      ((D(i+1,j)+D(i,j))/2 * u(i+1,j) - (D(i,j)+D(i-1,j))/2 * u(i,j)) / dx(i,j) + &
      ((D(i,j+1)+D(i,j))/2 * v(i+1,j) - (D(i,j)+D(i,j-1))/2 * v(i,j)) / dy(i,j) )
```

935

936 Figure 2. Implementation of Eq. (6) by basic operators. The *elf* and *elb* are the surface

937 elevations at times  $(t+1)$  and  $(t-1)$  respectively.

938



*\$ Equation (8)*

$$elf = elb - 2 * dt * ( DXF( AXB(D) * U ) + DYF( AYB(D) * V ) )$$

*\$ Equation (9)*

$$Uf = Db * Ub / Df - 2 * dt / Df * ( DXB( AXF( AXB(D) * U ) * AXF( U ) ) + DYF( AXB( AYB(D) * V ) * AYB( U ) ) - & \\ AXB( f * AYF( V ) * D ) + g * AXB( D ) * DXB( el ) - aam * AXB( D ) * ( DXB( DXF( Ub ) ) + DYF( DYB( Ub ) ) ) ) )$$

*\$ Equation (10)*

$$Vf = Db * Vb / Df - 2 * dt / Df * ( DXF( AYB( AXB(D) * U ) * AXB( V ) ) + DYB( AYF( AYB(D) * V ) * AYF( V ) ) + & \\ AYB( f * AXF( U ) * D ) + g * AYB( D ) * DYB( el ) - aam * AYB( D ) * ( DXF( DXB( Vb ) ) + DYB( DYF( Vb ) ) ) ) )$$

939

940 Figure 3. Implementation of the shallow water equations by basic operators. *elf*, *el* and

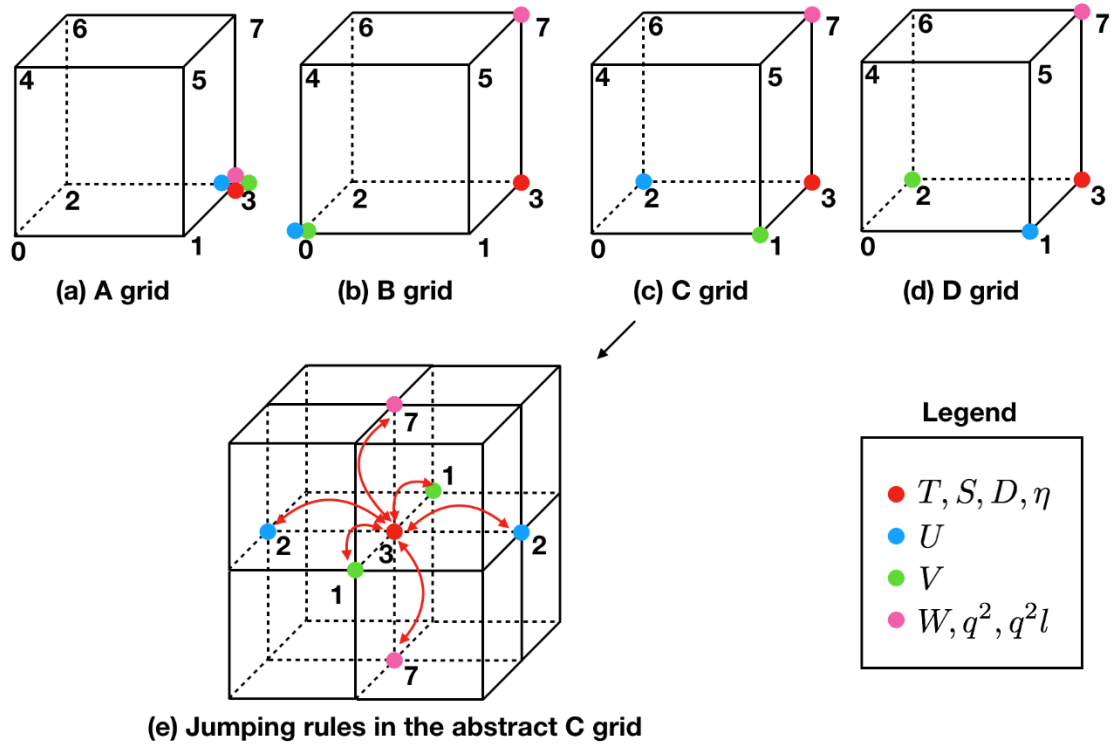
941 *elb* denote sea surface elevations at times  $(t+I)$ ,  $t$  and  $(t-I)$ , respectively. *Uf*, *U* and *Ub*

942 denote the zonal velocity at times  $(t+I)$ ,  $t$  and  $(t-I)$ , respectively. *Vf*, *V* and *Vb* denote

943 the meridional velocity at times  $(t+I)$ ,  $t$  and  $(t-I)$ , respectively. *aam* denotes the

944 viscosity coefficient.

945



946

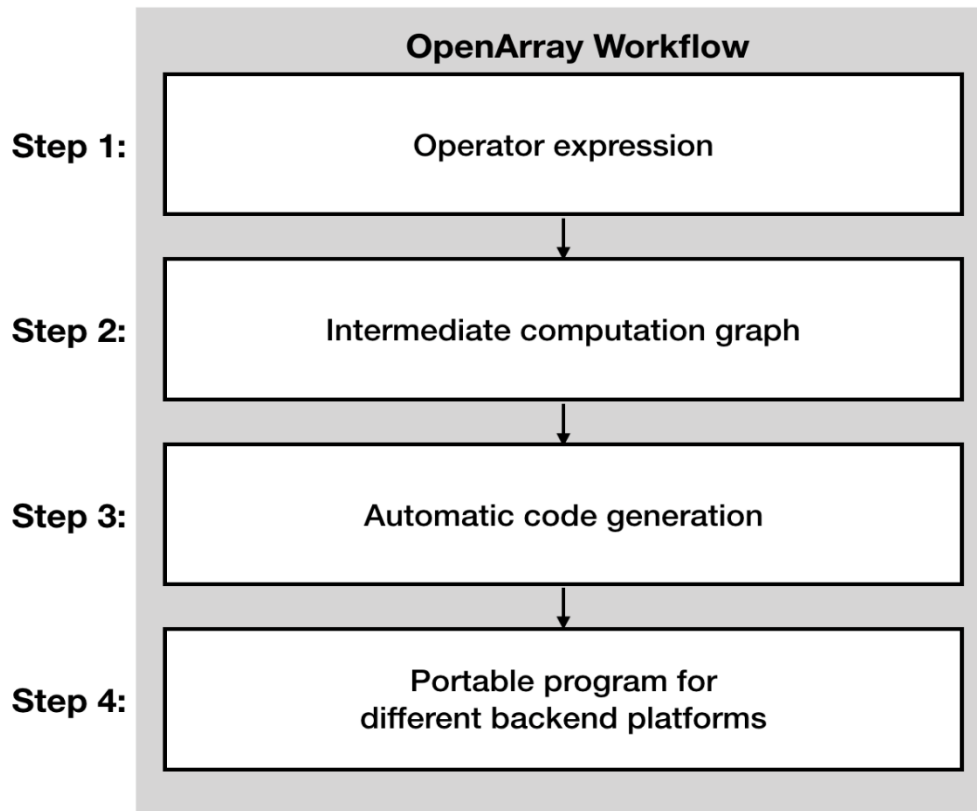
947

Figure 4. The schematic diagram of the relative positions of the variables on the

948

abstract staggered grid and the jumping procedures among the grid points.

949



950

951

952

Figure 5. The workflow of OpenArray.

<b>Formula</b>	$\eta_{t+1} = \eta_{t-1} - 2 * dt * \left( \delta_f^x(\bar{D}_b^x * U) + \delta_f^y(\bar{D}_b^y * V) \right)$
<b>Code</b>	$elf = elb - dt2 * (DXF(AXB(D)*U) + DYF(AYB(D)*V))$

953

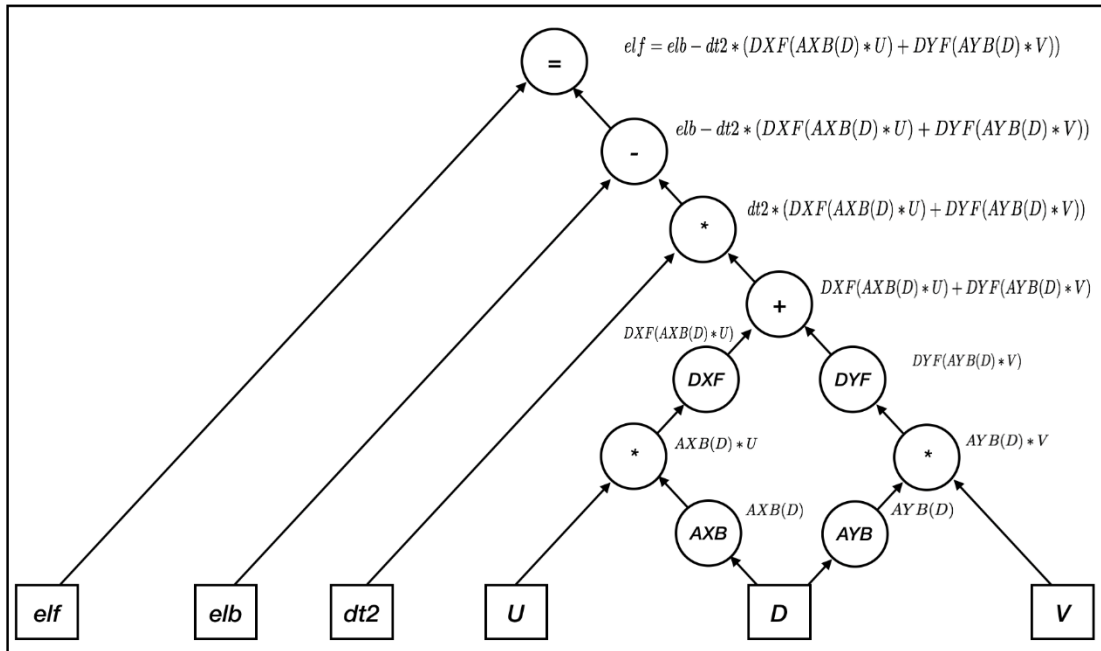
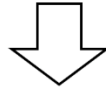
954

955

956

Figure 6. The effect of “The self-documenting code is the formula” illustrated by the sea surface elevation equation.

$$elf = elb - dt2 * (DXF(AXB(D) * U) + DYF(AYB(D) * V))$$



957

958

959

Figure 7. Parsing the operator expression form into the computation graph.

960  
961  
962

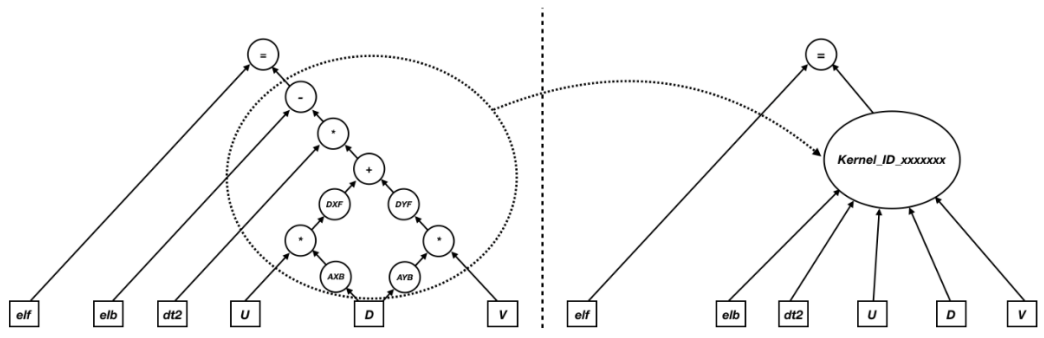
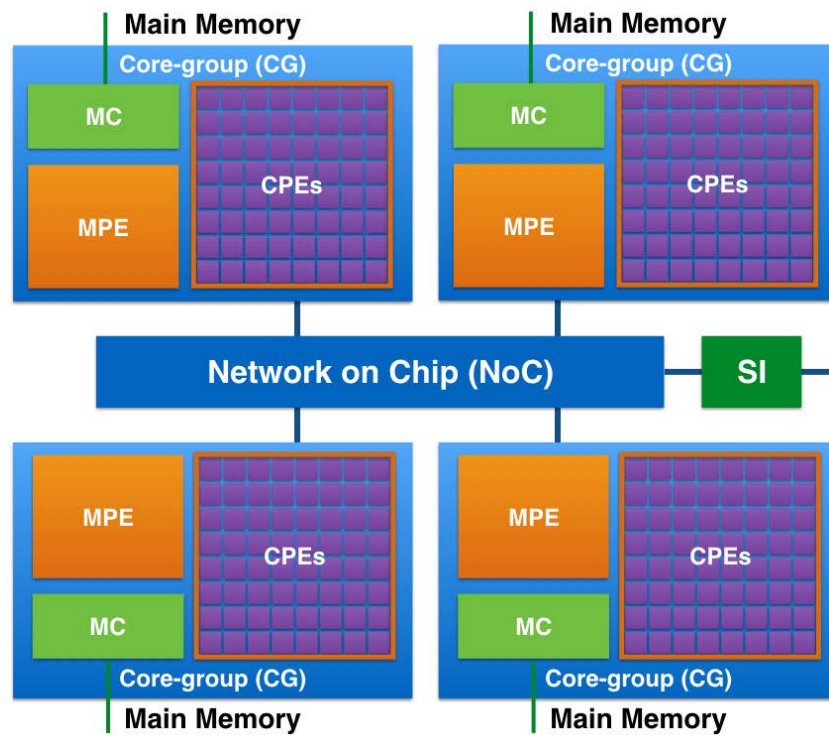


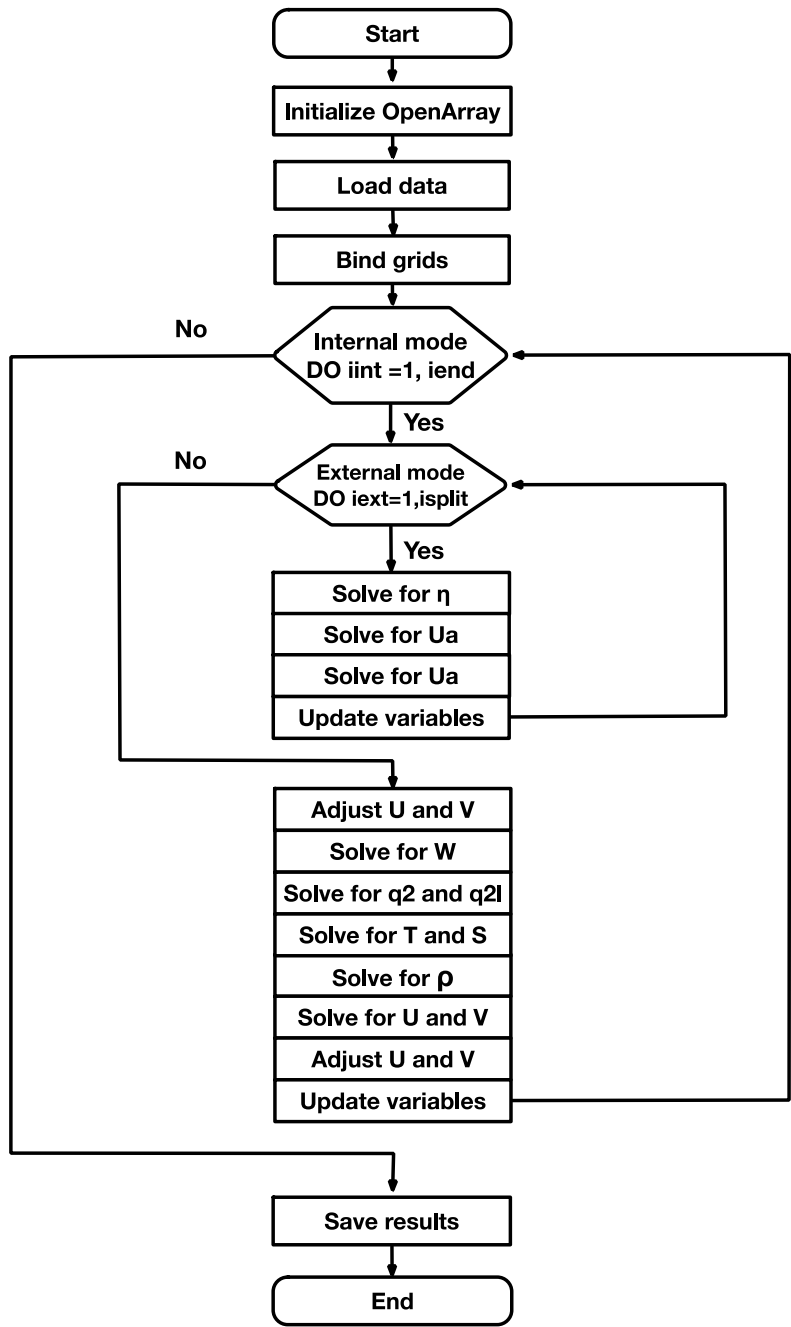
Figure 8. The schematic diagram of kernel fusion.



964

965 Figure 9. The MPE-CPEs hybrid architecture of the Sunway processor. Every Sunway  
 966 processor includes 4 Core-groups (CGs) connected by the Network on Chip (NoC).  
 967 Each CG consists of a management processing element (MPE), 64 computing  
 968 processing elements (CPEs) and a memory controller (MC). The Sunway processor  
 969 uses the system interface (SI) to connect with outside devices.

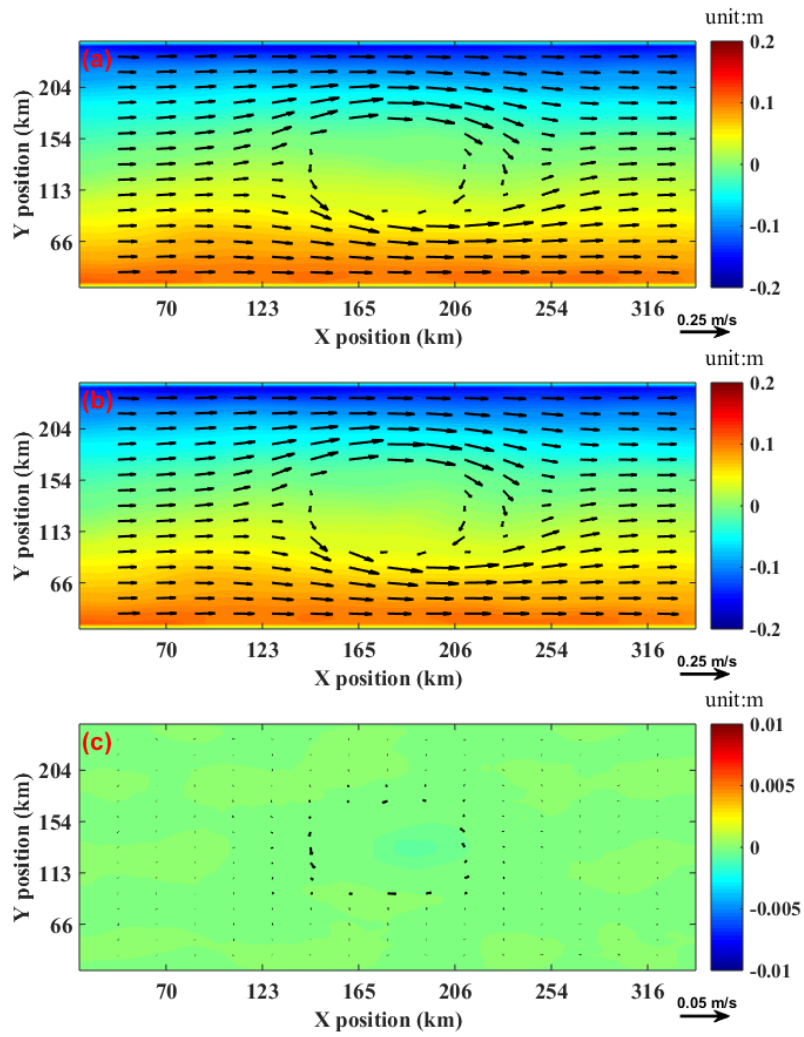
970



971  
972  
973

Figure 10. Flow diagram of GOMO





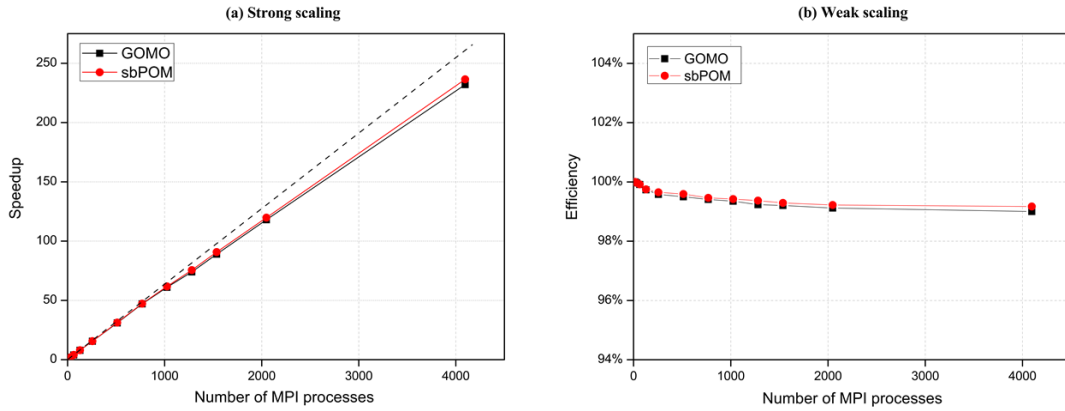
974

975 Figure 11. Comparison of the surface elevation (shaded) and currents at 3500 metres

976 depth (vector) between GOMO and sbPOM on the 4th model day. (a) GOMO, (b)

977 sbPOM, (c) GOMO-sbPOM.

978



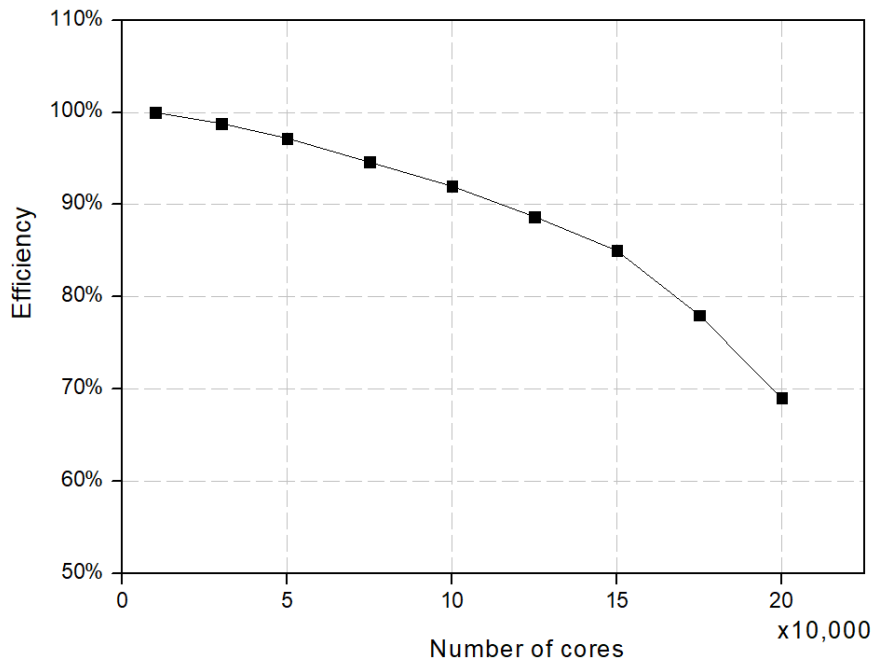
979

980 Figure 12. Performance comparison between sbPOM and GOMO on the X86 cluster.

981 (a) The strong scaling result; vertical axis denotes the speedup relative to 16 processes

982 in a single node. (b) The weak scaling result.

983



984

985

Figure 13. Parallel efficiency of GOMO on the Sunway platform.

986