OpenArray v1.0: A Simple Operator Library for the Decoupling of Ocean Modelling and Parallel Computing

Xiaomeng Huang[1,2,3], Xing Huang[1,3], Dong Wang[1,3], Qi Wu[1], Yi Li[3], Shixun Zhang[3], Yuwen Chen[1], Mingqing Wang[1,3], Yuan Gao[1], Qiang Tang[1], Yue Chen[1], Zheng Fang[1], Zhenya Song[2,4], Guangwen Yang[1,3]

[1] Ministry of Education Key Laboratory for Earth System Modeling, Department of Earth System Science, Tsinghua University, Beijing 100084, China
[2] Laboratory for Regional Oceanography and Numerical Modeling, Qingdao National Laboratory for Marine Science and Technology, Qingdao, 266237, China
[3] National Supercomputing Center in Wuxi, Wuxi, 214011, China
[4] First Institute of Oceanography, Ministry of Natural Resources, Qingdao, 266061, China

Corresponding author: hxm@tsinghua.edu.cn

Abstract

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

1. Introduction

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

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

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

63

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

Many efforts have been made to address the complexity of parallel programming for numerical simulations, such as operator overloading, source-to-source translator and domain specific language (DSL). Operator overloading supports the customized data type and provides simple operators and function interfaces to implement the model algorithm. This technique is widely used because the implementation is straightforward and easy to understand (Corliss and Griewank, 1994; Walther et al., 2003). However, it is prone to work inefficiently because overloading execution induces numerous unnecessary intermediate variables, consuming valuable memory bandwidth resources. Using a source-to-source translator offers another solution. As indicated by the name, this method converts one language, which is usually strictly constrained by self-defined rules, to another (Bae et al., 2013; Lidman et al., 2012). It requires tremendous work to develop and maintain a robust source-to-source compiler. Furthermore, DSLs can provide high-level abstraction interfaces that use mathematical notations similar to those used by domain scientists, so that they can write much more concise and more straightforward code. Some outstanding DSLs, such as ATMOL (van Engelen, 2001), ICON DSL (Torres et al., 2013), STELLA (Gysi et al., 2015) and ATLAS (Deconinck et al., 2017), are used by the numerical model community. Although they seem source-to-source technique, DSLs are newly-defined languages and produce executable programs instead of target languages. Therefore the new syntax makes it difficult for the modellers to master the DSLs. In addition, most DSLs are not yet supported by 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    $$\frac{\partial \eta}{\partial t} + \frac{\partial DU}{\partial x} + \frac{\partial DV}{\partial y} = 0,$$ (1)

159   where η 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 gird 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   *tmpD(i+1,j) = 0.5*(D(i+1,j)+D(i,j))*U(i+1,j),*   (2)

170   and

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

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

173

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

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

176    $\frac{0.5*\left(D(i,j+1)+D(i,j)\right)*V(i,j+1)-0.5*\left(D(i,j)+D(i,j-1)\right)*V(i,j)}{dy(i,j)^*} = 0,$ (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 η_{t+1} and η_{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 (δ_f^x, δ_b^y) and interpolation ($\overline{()}_f^x$, $\overline{()}_b^y$). The δ 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 $$\frac{\eta_{t+1}-\eta_{t-1}}{2*dt} + \delta_f^x(\overline{D}_b^x * U) + \delta_f^y(\overline{D}_b^y * V) = 0 \,. \tag{5}$$

195 Thus,

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

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 $$\frac{\partial \eta}{\partial t} + \frac{\partial DU}{\partial x} + \frac{\partial DV}{\partial y} = 0, \tag{7}$$

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

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

209 where f is the Coriolis parameter, g is the gravitational acceleration, and μ 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 $$\frac{\eta_{t+1}-\eta_{t-1}}{2*dt} + \delta_f^x(\overline{D}_b^x * U) + \delta_f^y(\overline{D}_b^y * V) = 0,$$ (10)

214 $$\frac{D_{t+1}U_{t+1}-D_{t-1}U_{t-1}}{2*dt} + \delta_b^x\left(\overline{\overline{D}_b^x * U}^x_f * \overline{U}_f^x\right) + \delta_f^y\left(\overline{\overline{D}_b^y * V}^x_b * \overline{U}_b^y\right) - f\,\overline{V}_f^y * D\,^x_b = -g *$$

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

216 $$\frac{D_{t+1}V_{t+1}-D_{t-1}V_{t-1}}{2*dt} + \delta_f^x\left(\overline{\overline{D}_b^x * U}^y_b * \overline{V}_b^x\right) + \delta_b^y\left(\overline{\overline{D}_b^y * V}^y_f * \overline{V}_f^y\right) + \overline{f\overline{U}_f^x * D}\,^y_b = -g *$$

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

271　$\frac{\eta_{t+1}(i,j)-\eta_{t-1}(i,j)}{2*dt} + \frac{0.25*(D(i+1,j)+D(i,j))*(U(i+1,j)+U(i+1,j+1))-0.25*(D(i,j)+D(i-1,j))*(U(i,j)+U(i,j+1))}{dx(i,j)^*} +$

272　$\frac{0.25*(D(i,j+1)+D(i,j))*(V(i,j+1)+V(i+1,j+1))-0.25*(D(i,j)+D(i,j-1))*(V(i,j)+V(i+1,j))}{dy(i,j)^*} = 0,$

273　(13)

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

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,

12

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 AYF 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

14

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

16

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-date (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 η 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, ρ), 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 σ directions, (T,

469   S, ρ) 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 seemly 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

5.2 Validation tests of GOMO

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

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

5.3 The weak and strong scalability of GOMO

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

553

5.4 Testing on the Sunway platform

We also test the scalability of GOMO on the Sunway platform. Supposing that the baseline is the runtime of GOMO at 10000 Sunway cores with a grid size of $4096\times4096\times50$, the parallel efficiency of GOMO can still reach 85% at 150000 cores, as shown in Fig. 13. However, we notice that the scalability declines sharply when the number of cores exceeds 150000. There are two reasons leading to this decline. First, the block size assigned to each core decreases as the number of cores increases, causing more communication during boundary region updating. Second, some processes cannot be accelerated even though more computing resources are available; for example, the time spent on creating the computation graph, generating the fusion kernels, and compiling the JIT cannot be reduced. Even though the fusion-kernel codes are generated and compiled only once at the beginning of a job, it consumes about 2 minutes. In a sense, OpenArray performs better when processing large-scale data, and GOMO is more suitable for high-resolution scenarios. In the future, we will further optimize the communication and graph-creating modules to improve the efficiency for large-scale cores.

6. Discussion

As we mentioned in Section 1, the advantages of OpenArray are easy-to-use, high efficiency and portability. Using OpenArray, the modellers without any parallel computing skill and experience can write simple operator expressions in Fortran to implement complex ocean models. The ocean models can be run on any CPU and Sunway platforms which have deployed the OpenArray library. We call this effect "write once, run everywhere". Other similar libraries (e.g., ATMOL, ICON DSL, and STELLA, COARRAY) require the users to manually control the boundary communication and task scheduling to some extent. In contrast, OpenArray implements completely implicit parallelism with user-friendly interfaces and programming 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
$$\frac{\partial \eta}{\partial t} + \frac{\partial UD}{\partial x} + \frac{\partial VD}{\partial y} + \frac{\partial W}{\partial \sigma} = 0, \tag{A1}$$

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

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

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

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

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

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

668
$$\rho = \rho(T, S, p),$$ (A6)

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

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

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

672
$$\left.\left.\left(\frac{\partial V}{\partial \sigma}\right)^2\right] + \frac{gE_3}{\rho_0}K_H\frac{\partial \rho}{\partial \sigma}\right\}\widetilde{W} - \frac{Dq^3}{B_1} + F_{q^2 l},$$ (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    $q^2 l$, respectivly. F_T and F_S are horizontal diffusion terms of T and S respectivly. $\widetilde{W}$

676   is the wall proximity function.

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

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

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

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

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

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

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

684   The equations governing the barotropic (external) mode in GOMO are obtained by

685   vertically integrating the baroclinic equations.

686
$$\frac{\partial \eta}{\partial t} + \frac{\partial U_A D}{\partial x} + \frac{\partial V_A D}{\partial y} = 0,$$ (A16)

687
$$\frac{\partial U_A D}{\partial t} + \frac{\partial (U_A)^2 D}{\partial x} + \frac{\partial U_A V_A D}{\partial y} - fV_A D + gD\frac{\partial \eta}{\partial x} = \widetilde{F}_{u_a} - wu(0) +$$

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

689
$$\frac{\partial V_A D}{\partial t} + \frac{\partial U_A V_A D}{\partial y} + \frac{\partial (V_A)^2 D}{\partial y} + fU_A D + gD\frac{\partial \eta}{\partial y} = \widetilde{F}_{v_a} - wv(0) +$$

690 $\quad wv(-1) - \frac{gD}{\rho_0} \int_{-1}^{0} \int_{\sigma}^{0} \left[D \frac{\partial \rho}{\partial y} - \frac{\partial D}{\partial y} \sigma' \frac{\partial \rho}{\partial \sigma} \right] d\sigma' d\sigma + G_{v_a},$ $\qquad$ (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 respectivly. G_{u_a} and G_{v_a} are the dispersion terms of U_A and V_A respectivly. The

694 subscript 'A' denotes vertical integration.

695

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

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

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

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

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

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

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

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

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

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

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

707 $\qquad AA_M = \int_{-1}^{0} (A_M) d\sigma.$ $\qquad$ (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 $\qquad \frac{\eta^{t+1} - \eta^{t-1}}{2dti} + \delta_f^x (\overline{D}_b^x U) + \delta_f^y (\overline{D}_b^y V) + \delta_f^z (W) = 0,$ $\qquad$ (B1)

713 $$\frac{(\overline{D}_b^x U)^{t+1}-(\overline{D}_b^x U)^{t-1}}{2dti} + \delta_b^x\left[\overline{(\overline{D}_b^x U)}_f^x \overline{U}_f^x\right] + \delta_f^y\left[\overline{(\overline{D}_b^y V)}_b^x \overline{U}_b^y\right] +$$

714 $$\delta_f^z(\overline{W}_b^x \overline{U}_b^z) - \overline{(\tilde{f}\overline{V}_f^y D)}_b^x - \overline{(f\overline{V}_f^y D)}_b^x + g\overline{D}_b^x \delta_b^x(\eta) = \delta_b^z\left[\frac{\overline{K_{Mb}}^x}{(\overline{D}_b^x)^{t+1}}\delta_f^z(U^{t+1})\right] +$$

715 $$\frac{g(\overline{D}_b^x)^2}{\rho_0}\int_\sigma^0\left[\delta_b^x(\overline{\rho}_b^z) - \frac{\sigma}{\overline{D}_b^x}\frac{\delta_b^x(D)}{}\delta_b^z(\overline{\rho}_b^x)\right]d\sigma' + F_u, \tag{B2}$$

716 $$\frac{(\overline{D}_b^y V)^{t+1}-(\overline{D}_b^y V)^{t-1}}{2dti} + \delta_f^x\left[\overline{(\overline{D}_b^x U)}_b^y \overline{V}_b^x\right] + \delta_b^y\left[\overline{(\overline{D}_b^y V)}_f^y \overline{V}_f^y\right] +$$

717 $$\delta_f^z(\overline{W}_b^y \overline{V}_b^z) + \overline{(\tilde{f}\overline{U}_f^x D)}_b^y + \overline{(f\overline{U}_f^x D)}_b^y + g\overline{D}_b^y \delta_b^y(\eta) = \delta_b^z\left[\frac{\overline{K_{Mb}}^y}{(\overline{D}_b^y)^{t+1}}\delta_f^z(V^{t+1})\right] +$$

718 $$\frac{g(\overline{D}_b^y)^2}{\rho_0}\int_\sigma^0\left[\delta_b^y(\overline{\rho}_b^z) - \frac{\sigma}{\overline{D}_b^y}\frac{\delta_b^y(D)}{}\delta_b^z(\overline{\rho}_b^y)\right]d\sigma' + F_v, \tag{B3}$$

719 $$\frac{(TD)^{t+1}-(TD)^{t-1}}{2dti} + \delta_f^x(\overline{T}_b^x U\overline{D}_b^x) + \delta_f^y(\overline{T}_b^y V\overline{D}_b^y) + \delta_f^z(\overline{T}_b^z W) =$$

720 $$\delta_b^z\left[\frac{K_H}{D^{t+1}}\delta_f^z(T^{t+1})\right] + F_T + \delta_f^z R, \tag{B4}$$

721 $$\frac{(SD)^{t+1}-(SD)^{t-1}}{2dti} + \delta_f^x(\overline{S}_b^x U\overline{D}_b^x) + \delta_f^y(\overline{S}_b^y V\overline{D}_b^y) + \delta_f^z(\overline{S}_b^z W) =$$

722 $$\delta_b^z\left[\frac{K_H}{D^{t+1}}\delta_f^z(S^{t+1})\right] + F_S, \tag{B5}$$

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

724 $$\frac{(q^2 D)^{t+1}-(q^2 D)^{t-1}}{2dti} + \delta_f^x(\overline{U}_b^z \overline{q^2}_b^x \overline{D}_b^x) + \delta_f^y(\overline{V}_b^z \overline{q^2}_b^y \overline{D}_b^y) +$$

725 $$\delta_f^z\overline{(Wq^2)}_b^z = \delta_b^z\left[\frac{\overline{K_{qf}}^z}{D^{t+1}}\delta_f^z(q^2)^{t+1}\right] + \frac{2K_M}{D}\left\{\left[\delta_b^z(\overline{U}_f^x)\right]^2 + \left[\delta_b^z(\overline{V}_f^y)\right]^2\right\} +$$

726 $$\frac{2g}{\rho_0}K_H\delta_b^z(\rho) - \frac{2Dq^3}{B_1 l} + F_{q^2}, \tag{B7}$$

727 $$\frac{(q^2 lD)^{t+1}-(q^2 lD)^{t-1}}{2dti} + \delta_f^x(\overline{U}_b^z \overline{q^2 l}_b^x \overline{D}_b^x) + \delta_f^y(\overline{V}_b^z \overline{q^2 l}_b^y \overline{D}_b^y) +$$

728 $$\delta_f^z\overline{(Wq^2 l)}_b^z = \delta_b^z\left[\frac{\overline{K_{qf}}^z}{D^{t+1}}\delta_f^z(q^2 l)^{t+1}\right] + lE_1\frac{K_M}{D}\left\{\left[\delta_b^z(\overline{U}_f^x)\right]^2 + \left[\delta_b^z(\overline{V}_f^y)\right]^2\right\}\widetilde{W} +$$

729 $$\frac{lE_1 E_3 g}{\rho_0}K_H\delta_b^z(\rho)\widetilde{W} - \frac{Dq^3}{B_1} + F_{q^2 l}, \tag{B8}$$

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

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

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

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

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

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

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

739

740 The discrete governing equations of barotropic (external) mode expressed by operators

741 are shown as below:

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

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

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

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

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

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

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

749

750 where

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

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

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

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

755

756

Appendix C: Descriptions of symbols

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

Table C1. Descriptions of symbols

Symbol	Description
η	Free surface elevation
H	Bottom topography
ua, va	Vertical average velocity in x, y direction, respectively
U, V, W	Velocity in x, y, σ direction, respectively
D	Fluid column depth
f	The Coriolis parameter
g	The gravitational acceleration
ρ_0	Constant density
ρ	Situ density
T	Potential temperature
S	Salinity
R	Surface solar radiation incident
$q^2/2$	Turbulence kinetic energy
l	Turbulence length scale
$q^2l/2$	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

30

778

779   **References**

780   Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat,

781   S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray,

782   D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y. and

783   Zheng, X.: TensorFlow: A System for Large-Scale Machine Learning, in 12th

784   {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI}

785   16), pp. 265–283, {USENIX} Association, Savannah, GA. [online] Available from:

786   https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi,

787   2016.

788   Alexander, K. and Easterbrook, S. M.: The software architecture of climate models: A

789   graphical comparison of CMIP5 and EMICAR5 configurations, Geosci. Model Dev.,

790   8(4), 1221–1232, doi:10.5194/gmd-8-1221-2015, 2015.

791 Arakawa, A. and Lamb, V. R.: A Potential Enstrophy and Energy Conserving Scheme

792   for the Shallow Water Equations, Mon. Weather Rev., doi:10.1175/1520-

793   0493(1981)109<0018:APEAEC>2.0.CO;2, 1981.

794 Bae, H., Mustafa, D., Lee, J. W., Aurangzeb, Lin, H., Dave, C., Eigenmann, R. and

795   Midkiff, S. P.: The Cetus source-to-source compiler infrastructure: Overview and

796   evaluation, in International Journal of Parallel Programming., 2013.

797 Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A.,

798   Bouchard, N., Warde-Farley, D. and Bengio, Y.: Theano: new features and speed

799   improvements, CoRR, abs/1211.5 [online] Available from:

800   http://arxiv.org/abs/1211.5590, 2012.

801 Beckmann, A. and Haidvogel, D. B.: Numerical simulation of flow around a tall

802   isolated seamount. Part I: problem formulation and model accuracy, J. Phys.

803   Oceanogr., 23(8), 1736–1753, doi:10.1175/1520-

804   0485(1993)023<1736:NSOFAA>2.0.CO;2, 1993.

805 Bloss, A., Hudak, P. and Young, J.: Code optimizations for lazy evaluation, Lisp Symb.

806   Comput., doi:10.1007/BF01806169, 1988.

807 Blumberg, A. F. and Mellor, G. L.: A description of a three-dimensional coastal ocean

808   circulation model, , (January 1987), 1–16, doi:10.1029/CO004p0001, 1987.

809 Bonan, G. B. and Doney, S. C.: Climate, ecosystems, and planetary futures: The

810   challenge to predict life in Earth system models, Science (80-.).,

811   doi:10.1126/science.aam8328, 2018.

812 Bretherton, C., Balaji, V. and Delworth, T. et al: A National Strategy for Advancing

813   Climate Modeling, National Academies Press., 2012.

814 Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S. H. and Skadron, K.:

815   Rodinia: A benchmark suite for heterogeneous computing, in Proceedings of the

816   2009 IEEE International Symposium on Workload Characterization, IISWC 2009.,

817   2009.

818 Chen, C., Liu, H. and Beardsley, R. C.: An unstructured grid, finite-volume, three-

819   dimensional, primitive equations ocean model: Application to coastal ocean and

820   estuaries, J. Atmos. Ocean. Technol., doi:10.1175/1520-

821   0426(2003)020<0159:AUGFVT>2.0.CO;2, 2003.

822   Collins, M., Minobe, S., Barreiro, M., Bordoni, S., Kaspi, Y., Kuwano-Yoshida, A.,

823   Keenlyside, N., Manzini, E., O'Reilly, C. H., Sutton, R., Xie, S. P. and Zolina, O.:

824   Challenges and opportunities for improved understanding of regional climate

825   dynamics, Nat. Clim. Chang., 8(2), 101–108, doi:10.1038/s41558-017-0059-8, 2018.

826   Corliss, G. and Griewank, A.: Operator Overloading as an Enabling Technology for

827   Automatic Differentiation, 1994.

828   Deconinck, W., Bauer, P., Diamantakis, M., Hamrud, M., Kühnlein, C., Maciel, P.,

829   Mengaldo, G., Quintino, T., Raoult, B., Smolarkiewicz, P. K. and Wedi, N. P.: Atlas :

830   A library for numerical weather prediction and climate modelling, Comput. Phys.

831   Commun., 220, 188–204, doi:10.1016/j.cpc.2017.07.006, 2017.

832   Dennis, J. M.: Inverse space-filling curve partitioning of a global ocean model, Proc. -

833   21st Int. Parallel Distrib. Process. Symp. IPDPS 2007; Abstr. CD-ROM, 1–10,

834   doi:10.1109/IPDPS.2007.370215, 2007.

835   van Engelen, R. a.: ATMOL: A Domain-Specific Language for Atmospheric Modeling,

836   J. Comput. Inf. Technol., 9(4), 289–303, doi:10.2498/cit.2001.04.02, 2001.

837   Frigo, M. and Strumpen, V.: Cache oblivious stencil computations, , 361,

838   doi:10.1145/1088149.1088197, 2005.

839   Fu, H., He, C., Chen, B., Yin, Z., Zhang, Z., Zhang, W., Zhang, T., Xue, W., Liu, W.,

840   Yin, W. and others: 18.9-Pflops nonlinear earthquake simulation on Sunway

841   TaihuLight: enabling depiction of 18-Hz and 8-meter scenarios, in Proceedings of

842   the International Conference for High Performance Computing, Networking, Storage

843   and Analysis., 2017.

844   Griffies, S. M., Böning, C., Bryan, F. O., Chassignet, E. P., Gerdes, R., Hasumi, H.,

845   Hirst, A., Treguier, A.-M. and Webb, D.: Developments in ocean climate modelling,

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

847   Gysi, T., Osuna, C., Fuhrer, O., Bianco, M. and Schulthess, T. C.: STELLA: A Domain-

848   specific Tool for Structured Grid Methods in Weather and Climate Models, Proc. Int.

849   Conf. High Perform. Comput. Networking, Storage Anal. - SC '15, 1–12,

850   doi:10.1145/2807591.2807627, 2015.

851   Huang, W., Ghosh, S., Velusamy, S., Sankaranarayanan, K., Skadron, K. and Stan, M.

852   R.: HotSpot: A compact thermal modeling methodology for early-stage VLSI design,

853   IEEE Trans. Very Large Scale Integr. Syst., doi:10.1109/TVLSI.2006.876103, 2006.

854   Huang, X. M., Wang, W. C., Fu, H. H., Yang, G. W., Wang, B. and Zhang, C.: A fast

855   input/output library for high-resolution climate models, Geosci. Model Dev., 7(1),

856   93–103, doi:10.5194/gmd-7-93-2014, 2014.

857   Korn, P.: Formulation of an unstructured grid model for global ocean dynamics, J.

858   Comput. Phys., 339, 525–552, doi:10.1016/j.jcp.2017.03.009, 2017.

859   Lawrence, B. N., Rezny, M., Budich, R., Bauer, P., Behrens, J., Carter, M., Deconinck,

860   W., Ford, R., Maynard, C., Mullerworth, S., Osuna, C., Porter, A., Serradell, K.,

861   Valcke, S., Wedi, N. and Wilson, S.: Crossing the chasm: How to develop weather

862   and climate models for next generation computers?, Geosci. Model Dev.,

863   doi:10.5194/gmd-11-1799-2018, 2018.

864   Levin, J. G., Iskandarani, M. and Haidvogel, D. B.: A nonconforming spectral element

865   ocean model, Int. J. Numer. Methods Fluids, 34(6), 495–525, doi:10.1002/1097-

866   0363(20001130)34:6<495::AID-FLD68>3.0.CO;2-K, 2000.

867   Lidman, J., Quinlan, D. J., Liao, C. and McKee, S. A.: ROSE::FTTransform - A source-

868   to-source translation framework for exascale fault-tolerance research, Proc. Int. Conf.

869   Dependable Syst. Networks, (June), doi:10.1109/DSNW.2012.6264672, 2012.

870   Mellor-Crummey, J., Adhianto, L., Scherer III, W. N. and Jin, G.: A New Vision for

871   Coarray Fortran, in Proceedings of the Third Conference on Partitioned Global

872   Address Space Programing Models, p. 5:1--5:9, ACM, New York, NY, USA., 2009.

873   Mellor, G. L.: Users guide for a three-dimensional, primitive equation, numerical ocean

874   model (June 2003 version), Prog. Atmos. Ocean. Sci, Princet. Univ., (October), 53,

875   2003.

876   Mellor, G. L. and Yamada, T.: Development of a turbulence closure model for

geophysical fluid problems, Rev. Geophys., doi:10.1029/RG020i004p00851, 1982.

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

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

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

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

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

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

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

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

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

Walther, A., Griewank, A. and Vogel, O.: ADOL-C: Automatic Differentiation Using Operator Overloading in C++, PAMM, doi:10.1002/pamm.200310011, 2003.

906   Xu, S., Huang, X., Oey, L. Y., Xu, F., Fu, H., Zhang, Y. and Yang, G.: POM.GPU-v1.0:

907   A GPU-based princeton ocean model, Geosci. Model Dev., doi:10.5194/gmd-8-

908   2815-2015, 2015.

909

910 **Tables**

911 Table 1. Definitions of the twelve basic operators

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

912

913

914

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

The initial position of *var*	The position of [A/D]*X*[F/B] (var)	The position of [A/D]*Y*[F/B] (var)	The position of [A/D]*Z*[F/B] (var)
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

915

916

917

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

918

919

920

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

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

921

922

923

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

924

925

926 **Figures**

927
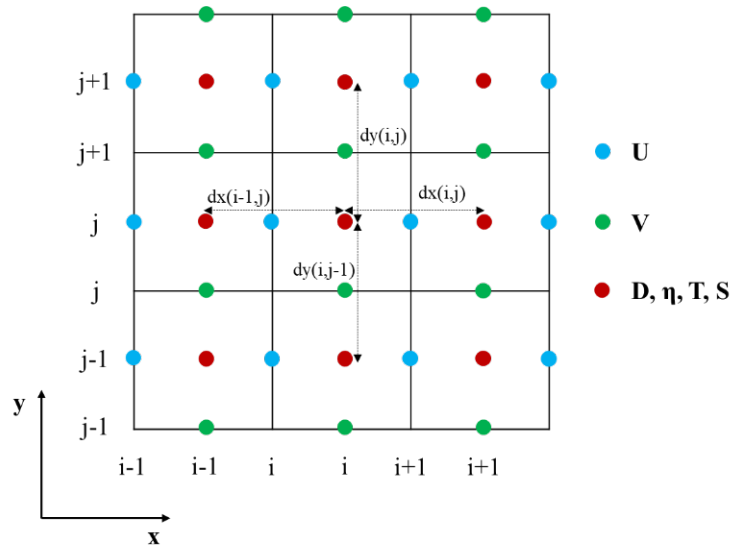
928
929 Figure 1. Arrangement of variables in the staggered Arakawa C grid.
930

931

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

$ 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))
```

932

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

934   elevations at times (*t+1*) and (*t-1*) respectively.

935

$ 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))))

936

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

938 *elb* denote sea surface elevations at times $(t+1)$, t and $(t-1)$, respectively. *Uf, U* and *Ub*

939 denote the zonal velocity at times $(t+1)$, t and $(t-1)$, respectively. *Vf*, *V* and *Vb* denote

940 the meridional velocity at times $(t+1)$, t and $(t-1)$, respectively. *aam* denotes the

941 viscosity coefficient.

942

(a) A grid (b) B grid (c) C grid (d) D grid

(e) Jumping rules in the abstract C grid

Legend

- T, S, D, η
- U
- V
- $W, q^2, q^2 l$
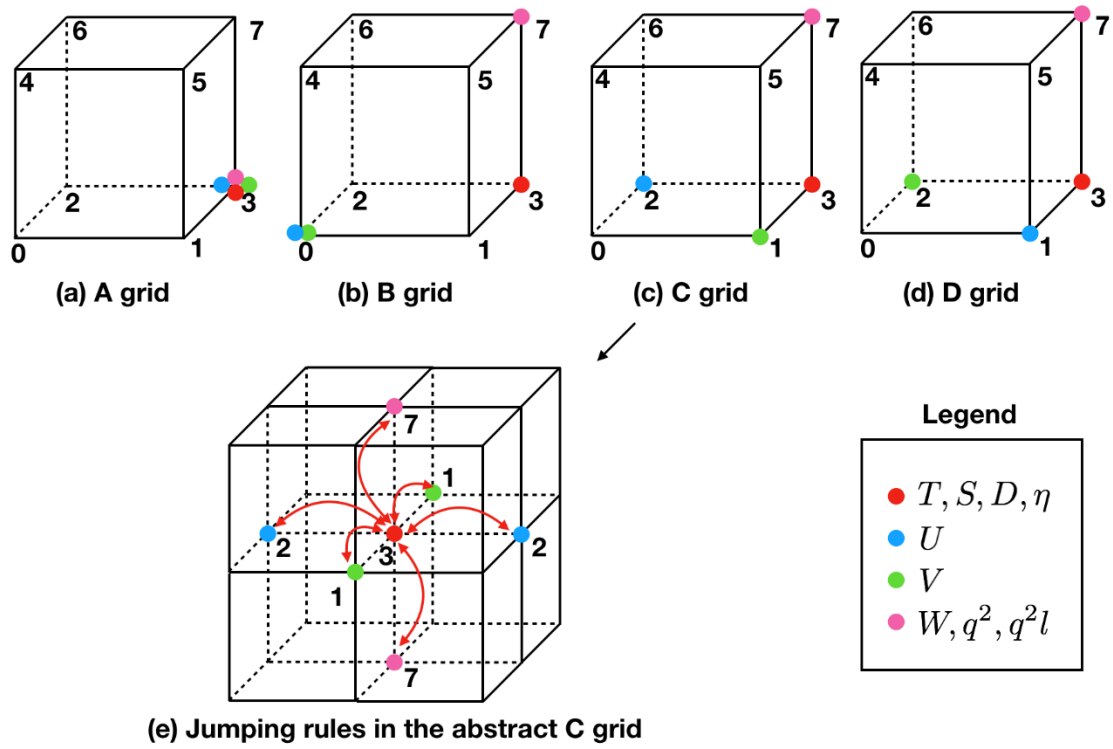
Figure 4. The schematic diagram of the relative positions of the variables on the abstract staggered grid and the jumping procedures among the grid points.

OpenArray Workflow

Step 1: Operator expression

Step 2: Intermediate computation graph

Step 3: Automatic code generation

Step 4: Portable program for different backend platforms

947
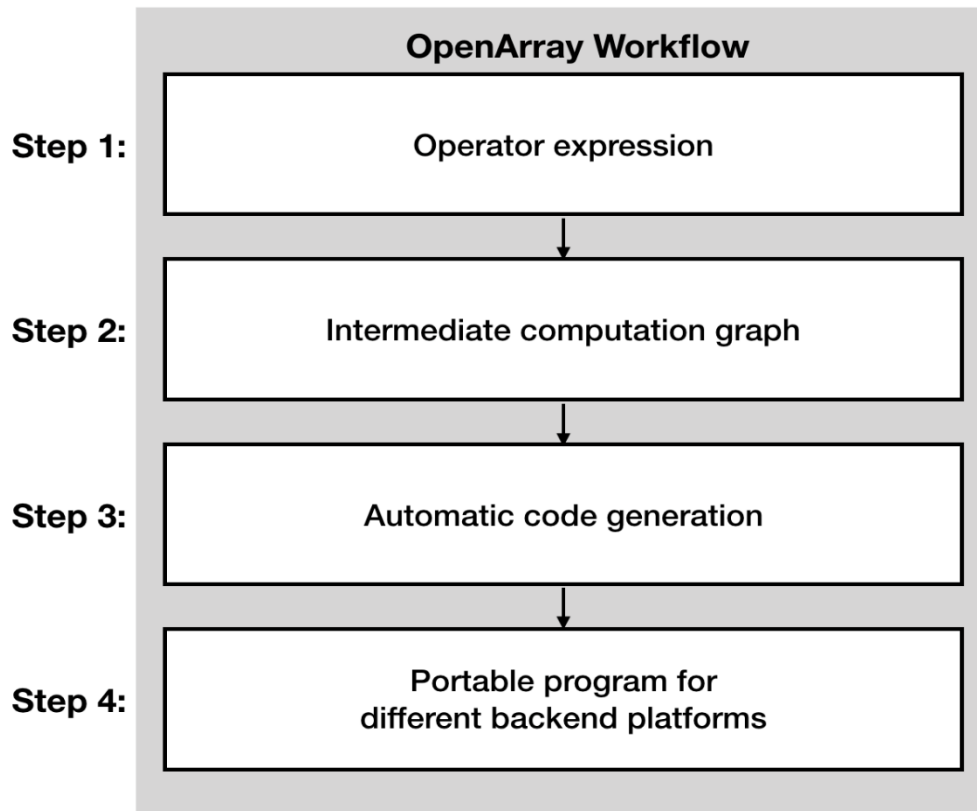
948   Figure 5. The workflow of OpenArray.

949

Formula

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

Code

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

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

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

954
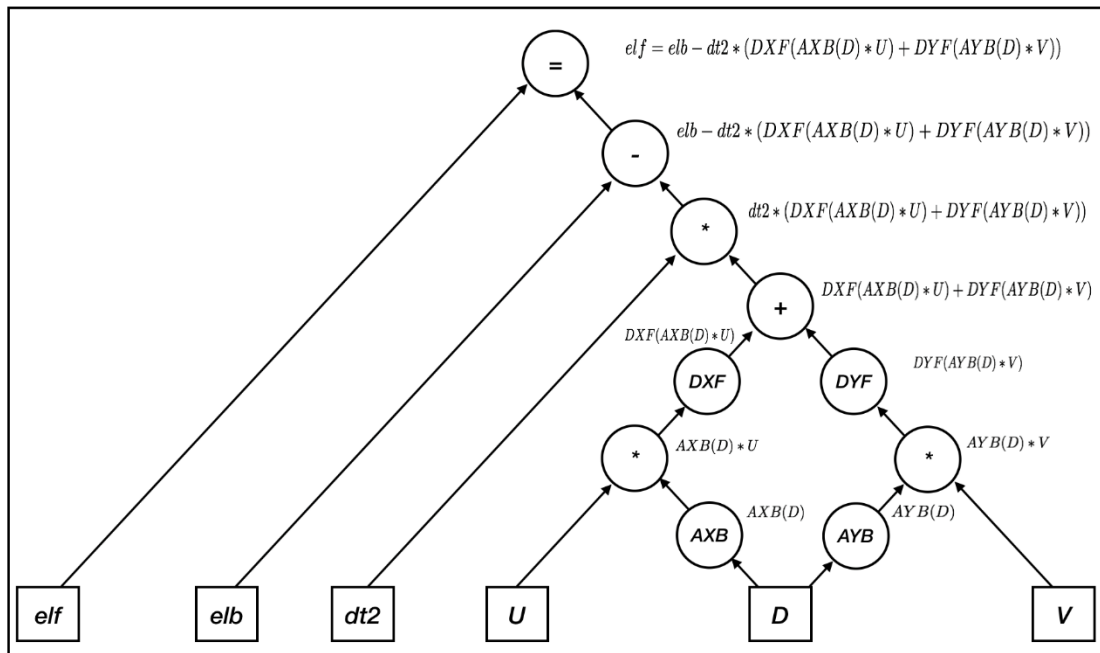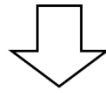
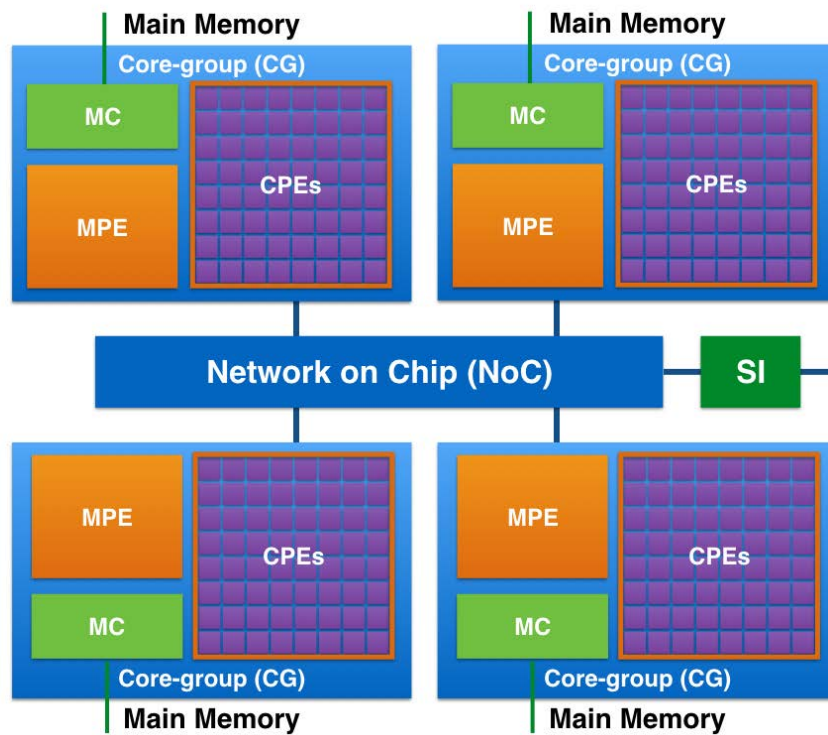955   Figure 7. Parsing the operator expression form into the computation graph.

956

957
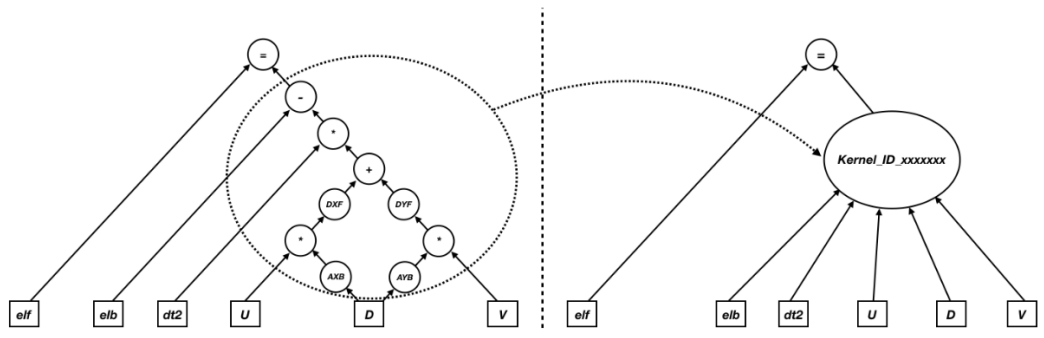
958   Figure 8. The schematic diagram of kernel fusion.

959

960

961

962   Figure 9. The MPE-CPEs hybrid architecture of the Sunway processor. Every Sunway

963   processor includes 4 Core-groups (CGs) connected by the Network on Chip (NoC).

964   Each CG consists of a management processing element (MPE), 64 computing

965   processing elements (CPEs) and a memory controller (MC). The Sunway processor
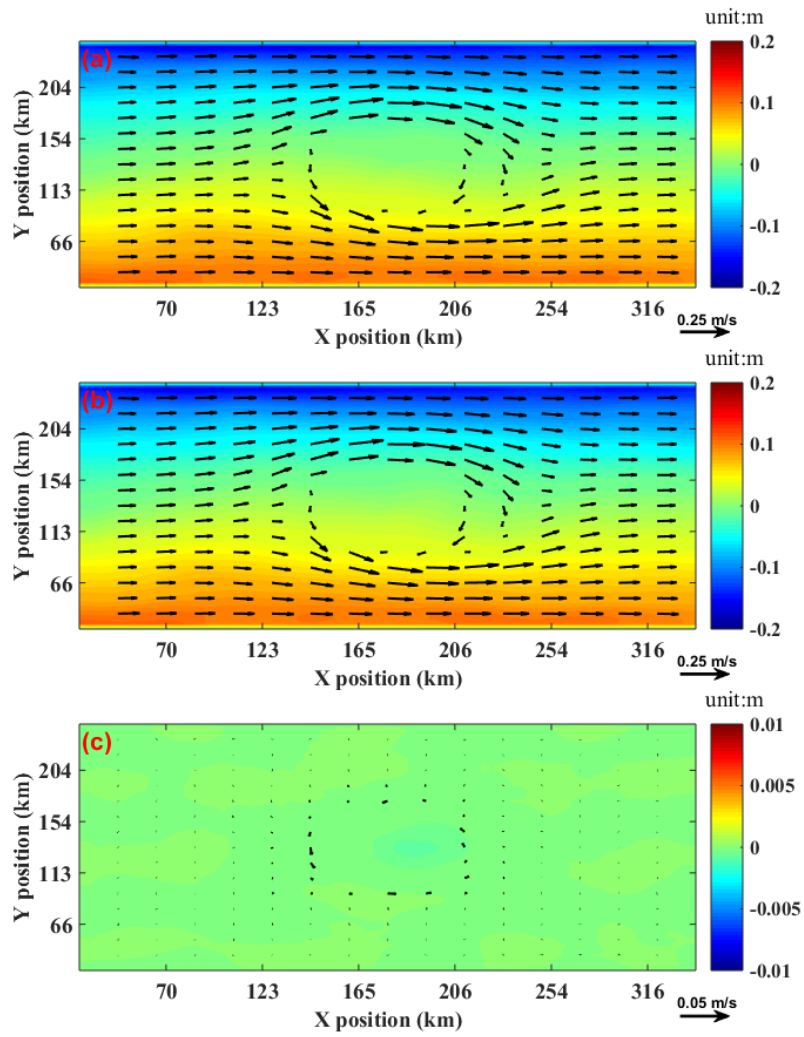
966   uses the system interface (SI) to connect with outside devices.

967

968
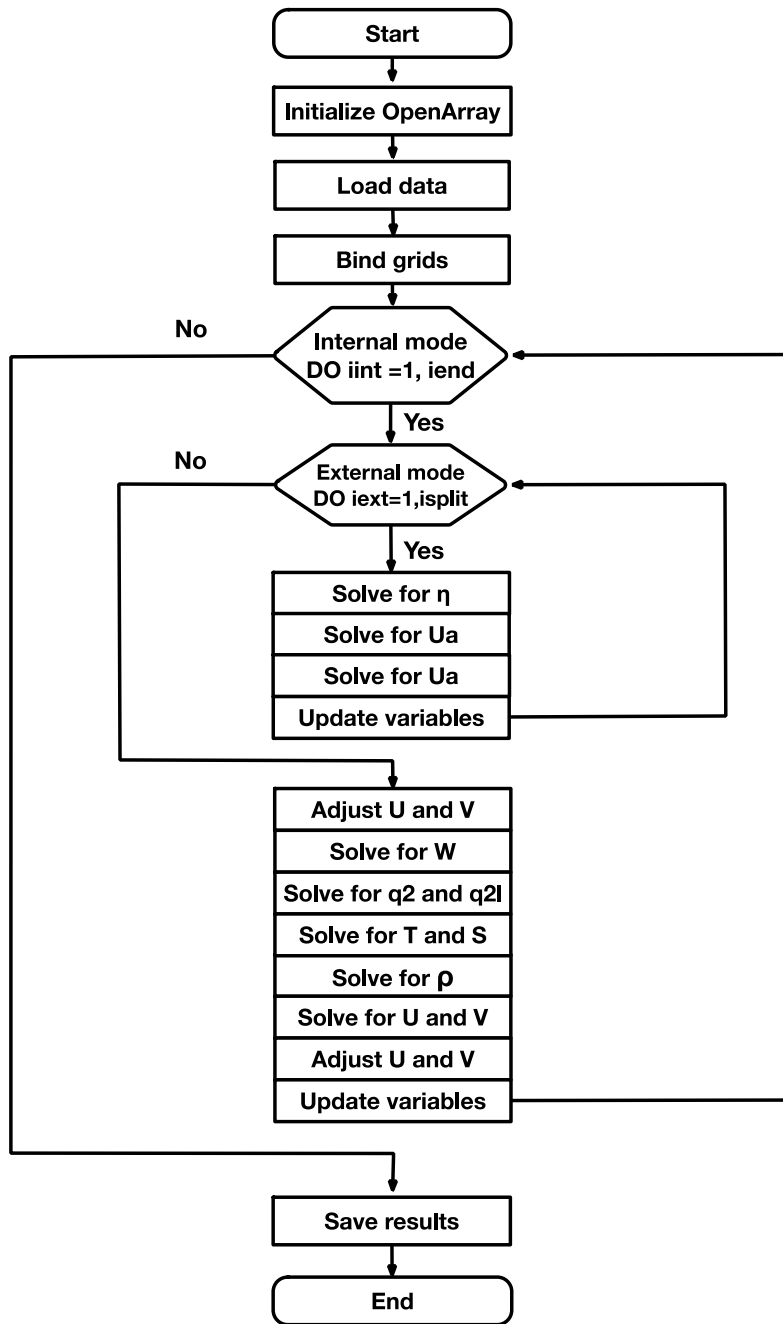
969   Figure 10. Flow diagram of GOMO
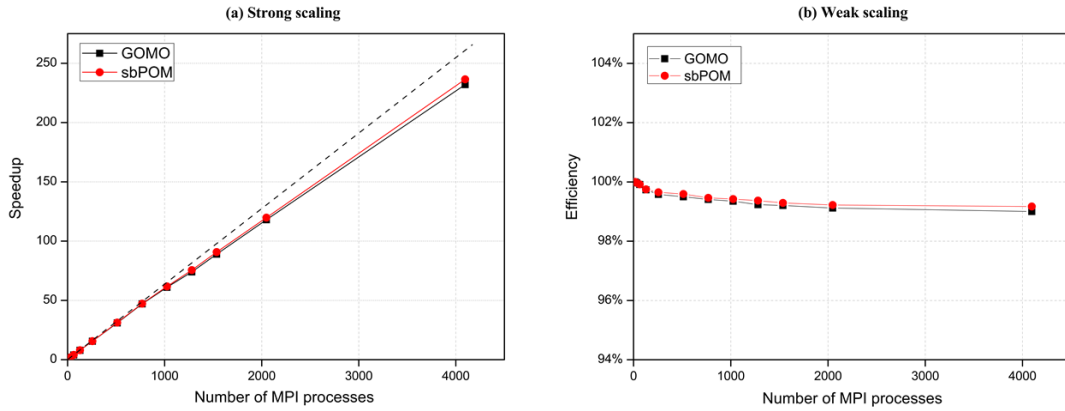
970

971

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

973　depth (vector) between GOMO and sbPOM on the 4th model day. (a) GOMO, (b)
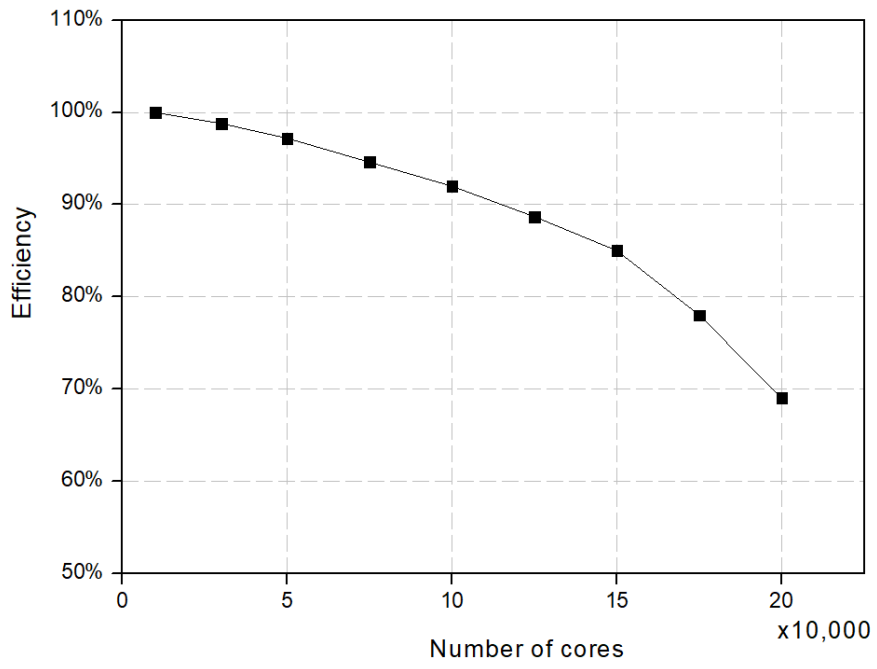
974　sbPOM, (c) GOMO-sbPOM.

975

976

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

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

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

980

981

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

983