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

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

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.

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

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

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

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

supported platforms.

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

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

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

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

2.1 Array

To achieve this simplicity, we designed a derived data type, *Array*, which inspired our project name, OpenArray. The new *Array* data type comprises a series of information, including a 3-dimensional (3D) array to store data, a pointer to the computational grid, a Message Passing Interface (MPI) communicator, the size of the halo region and other information about the data distribution. All the information is used to manipulate the *Array* as an object to simplify the parallel computing. In traditional ocean models, calculations for each grid point and the *i*, *j*, and *k* loops in the horizontal and vertical directions are unavoidable. The advantage of taking the *Array* as an object is the significant reduction in the number of loop operations in the models, making the code more intuitive and readable. When using the OpenArray library in a program, one can use *type(Array)* to declare new variables.

2.2 Operator

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

$$\frac{\partial \eta}{\partial t} + \frac{\partial DU}{\partial x} + \frac{\partial DV}{\partial y} = 0, \tag{1}$$

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

$$tmpD(i+1,j) = 0.5*(D(i+1,j)+D(i,j))*U(i+1,j), \tag{2}$$

and

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

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

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

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

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

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)),* subscripts η_{t+1} and η_{t-1} denote the surface elevations at the (*t+1*) time step and (*t-1*) time step. To simplify the discrete form, we introduce some notation for the differentiation (δ_f^x , δ_b^y) and interpolation ($\overline{(\,)}_f^x$, $\overline{(\,)}_b^y$). The δ and overbar symbols define the differential operator and average operator. The subscript *x* or *y* denotes that the operation acts in the *x* or *y* direction, and the superscript *f* or *b* denotes that the approximation operation is forward or backward.

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

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

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

Thus,

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

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

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

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

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

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

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

$$\frac{\eta_{t+1}-\eta_{t-1}}{2*dt}+\delta_f^x(\,\overline{D}_b^x*U)+\delta_f^y\left(\overline{D}_b^y*V\right)=0, \tag{10}$$

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

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

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

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

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

2.3 Abstract staggered grid

Most ocean models are implemented based on the staggered Arakawa grids (Arakawa and Lamb, 1981; Griffies et al., 2000). The variables in ocean models are allocated at different grid points. The calculations that use these variables are performed after several reasonable interpolations or differences. When we call the differential operations on a staggered grid, the difference value between adjacent points should be divided by the grid increment to obtain the final result. Setting the correct grid increment for modellers is troublesome work that is extremely prone to error, especially when the grid is nonuniform. Therefore, we propose an abstract staggered grid to support flexible switching of operator calculations among different staggered grids. When the grid information is provided at the initialization phase of OpenArray, a set of grid increments, including horizontal increments (*dx(i,j), dy(i,j)*) and vertical increment (*dz(k)*), will be combined with each corresponding physical variable through grid binding. Thus, the operators can implicitly set the correct grid increments for different *Array* variables, even if the grid is nonuniform.

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

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

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

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

$$\frac{\eta_{t+1}(i,j)-\eta_{t-1}(i,j)}{2*dt}+\frac{0.25*\left(D(i+1,j)+D(i,j)\right)*(U(i+1,j)+U(i+1,j+1))-0.25*\left(D(i,j)+D(i-1,j)\right)*(U(i,j)+U(i,j+1))}{dx(i,j)^*}+$$
$$\frac{0.25*\left(D(i,j+1)+D(i,j)\right)*(V(i,j+1)+V(i+1,j+1))-0.25*\left(D(i,j)+D(i,j-1)\right)*(V(i,j)+V(i+1,j))}{dy(i,j)^*}=0, \tag{13}$$

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

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

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

3. Design of OpenArray

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

3.1 Operator expression

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

Using these operators to express the discrete PDE equation, the code and formula are very similar. We call this effect “the self-documenting code is the formula”. Fig. 6 shows the one-to-one correspondence of each item in the code and the items in the sea surface elevation equation. The code is very easy to program and understand. Clearly, the basic operators and the combined operators greatly simplify the development and maintenance of ocean models. The complicated parallel and optimization techniques are hidden behind these operators. Modellers no longer need to care about details and can escape from the “parallelism swamp”, and can therefore concentrate on the scientific issues.

3.2 Intermediate computation graph

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

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

notation is the addition function, DXF and DYF are the differential functions, and AXF and AYF are the average functions.

3.3 Code generation

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

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

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

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

Therefore, we implemented a general boundary management module to implicitly maintain and update the local boundary information so that the modellers no longer need to address the message communication. The boundary management module uses asynchronous communication to update and maintain the data of the boundary region, which is useful for simultaneous computing and communication. These procedures of asynchronous communication are implicitly invoked when calling the basic kernel or the fused kernel to ensure that the parallel details are completely transparent to the modellers. For the global boundary conditions of the limited physical domains, the values at the physical border are always set to zero within the operators and operator expressions. In realistic cases, the global boundary conditions are set by a series of functions (e.g., radiation, wall) provided by OpenArray.

3.4 Portable program for different backend platforms

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

According to the TOP500 list released in November 2018, the Sunway TaihuLight is ranked third in the world, with a LINPACK benchmark rating of 93 Petaflops provided by Sunway many-core processors (or Sunway CPUs). As shown in Fig. 9, every Sunway CPU includes 260 processing elements (or cores) that are divided into 4 core-groups. Each core-group consists of 64 computing processing elements (CPEs) and a management processing element (MPE) (Qiao et al., 2017). CPEs handle large-scale computing tasks and MPE is responsible for the task scheduling and communication. The relationship between MPE and CPE is like that between CPU and many-core accelerator, except for they are fused into a single Sunway processor sharing a unified memory space. To make the most of the computing resources of the Sunway TaihuLight, we generate kernel functions for the MPE, which is responsible for the thread control, and CPE, which performs the computations. The kernel functions are fully optimized with several code optimization techniques (Pugh, 1991) such as loop tiling, loop aligning, single-instruction multiple-date (SIMD) vectorization, and function inline. In addition, due to the high memory access latency of CPEs, we accelerate data access by providing instructions for direct memory access in the kernel to transfer data between the main memory and local memory (Fu et al., 2017).

4. Implementation of GOMO

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

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

Figure 10 shows the basic flow diagram of GOMO. At the beginning, we initialize OpenArray to make all operators suitable for GOMO. After loading the initial values and the model parameters, the distance information is input into the differential operators through grid binding. In the external mode, the main consumption is computing the 2D sea surface elevation η and column-averaged velocity (Ua, Va). In the internal mode, 3D array computations predominate in order to calculate baroclinic motions (*U, V, W*), tracers (*T, S, ρ*), and turbulence closure scheme (q^2, q^2l) (Mellor and Yamada, 1982), where (*U, V, W*) are the velocity fields in the *x*, *y* and σ directions, (*T, S, ρ*) are the potential temperature, the salinity and the density. ($q^2/2$, $q^2l/2$) are the turbulence kinetic energy and production of turbulence kinetic energy with turbulence length scale.

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

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

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

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

5. Results

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

5.1 Benchmark testing

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

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.

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 4096×4096×50, 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.

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

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

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

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

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

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

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

7. Conclusion

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

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

Appendix A: Continuous governing equations

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

$$\frac{\partial \eta}{\partial t}+\frac{\partial UD}{\partial x}+\frac{\partial VD}{\partial y}+\frac{\partial W}{\partial \sigma}=0, \tag{A1}$$

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

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

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

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

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

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

$$\rho=\rho(T,S,p), \tag{A6}$$

$$\frac{\partial q^2D}{\partial t}+\frac{\partial Uq^2D}{\partial x}+\frac{\partial Vq^2D}{\partial y}+\frac{\partial Wq^2}{\partial \sigma}=\frac{\partial}{\partial \sigma}\left(\frac{K_q}{D}\frac{\partial q^2}{\partial \sigma}\right)+\frac{2K_M}{D}\left[\left(\frac{\partial U}{\partial \sigma}\right)^2+\left(\frac{\partial V}{\partial \sigma}\right)^2\right]+\frac{2g}{\rho_0}K_H\frac{\partial \rho}{\partial \sigma}-\frac{2Dq^3}{B_1 l}+F_{q^2}, \tag{A7}$$

$$\frac{\partial q^2lD}{\partial t}+\frac{\partial Uq^2lD}{\partial x}+\frac{\partial Vq^2lD}{\partial y}+\frac{\partial Wq^2l}{\partial \sigma}=\frac{\partial}{\partial \sigma}\left(\frac{K_q}{D}\frac{\partial q^2l}{\partial \sigma}\right)+E_1l\left\{\frac{K_M}{D}\left[\left(\frac{\partial U}{\partial \sigma}\right)^2+\left(\frac{\partial V}{\partial \sigma}\right)^2\right]+\frac{gE_3}{\rho_0}K_H\frac{\partial \rho}{\partial \sigma}\right\}\widetilde{W}-\frac{Dq^3}{B_1}+F_{q^2l}, \tag{A8}$$

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

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

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

$$F_T=\frac{\partial}{\partial x}(A_HH\frac{\partial T}{\partial x})+\frac{\partial}{\partial y}(A_HH\frac{\partial T}{\partial y}), \tag{A11}$$

$$F_S=\frac{\partial}{\partial x}(A_HH\frac{\partial S}{\partial x})+\frac{\partial}{\partial y}(A_HH\frac{\partial S}{\partial y}), \tag{A12}$$

$$F_{q^2}=\frac{\partial}{\partial x}(A_MH\frac{\partial q^2}{\partial x})+\frac{\partial}{\partial y}(A_MH\frac{\partial q^2}{\partial y}), \tag{A13}$$

$$F_{q^2l}=\frac{\partial}{\partial x}(A_MH\frac{\partial q^2l}{\partial x})+\frac{\partial}{\partial y}(A_MH\frac{\partial q^2l}{\partial y}), \tag{A14}$$

$$\widetilde{W}=1+\frac{E_2l}{\kappa}\left(\frac{1}{\eta-z}+\frac{1}{\mathrm{H}-z}\right). \tag{A15}$$

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

$$\frac{\partial \eta}{\partial t}+\frac{\partial U_AD}{\partial x}+\frac{\partial V_AD}{\partial y}=0, \tag{A16}$$

$$\frac{\partial U_AD}{\partial t}+\frac{\partial (U_A)^2D}{\partial x}+\frac{\partial U_AV_AD}{\partial y}-fV_AD+gD\frac{\partial \eta}{\partial x}=\tilde{F}_{u_a}-wu(0)+wu(-1)-\frac{gD}{\rho_0}\int_{-1}^{0}\int_{\sigma}^{0}\left[D\frac{\partial \rho}{\partial x}-\frac{\partial D}{\partial x}\sigma'\frac{\partial \rho}{\partial \sigma}\right]d\sigma' d\sigma+G_{u_a}, \tag{A17}$$

$$\frac{\partial V_AD}{\partial t}+\frac{\partial U_AV_AD}{\partial y}+\frac{\partial (V_A)^2D}{\partial y}+fU_AD+gD\frac{\partial \eta}{\partial y}=\tilde{F}_{v_a}-wv(0)+$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Appendix B: Discrete governing equations

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{2g}{\rho_0}K_H\delta_b^z(\rho)-\frac{2Dq^3}{B_1l}+F_{q^2}, \tag{B7}$$

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

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

$$\frac{lE_1E_3g}{\rho_0}K_H\delta_b^z(\rho)\widetilde{W}-\frac{Dq^3}{B_1}+F_{q^2l}, \tag{B8}$$

where F_u, F_v, F_{q^2}, and F_{q^2l} are horizontal kinematic viscosity terms of u, v, q^2, and q^2l, respectivly. F_T and F_S are horizontal diffusion terms of T and S respectivly.

$$F_u = \delta_b^x\left[2A_M D\delta_f^x(U^{t-1})\right] + \delta_f^y\left\{\overline{\left(\overline{A_M}^{x}_{b}\right)_b}^{y}\,\overline{\left(\overline{D}^{x}_{b}\right)_b}^{y}\left[\delta_b^x(V)^{t-1} + \delta_b^y(U)^{t-1}\right]\right\}, \tag{B9}$$

$$F_v = \delta_b^y\left[2A_M D\delta_f^y(V^{t-1})\right] + \delta_f^x\left\{\overline{\left(\overline{A_M}^{x}_{b}\right)_b}^{y}\,\overline{\left(\overline{D}^{x}_{b}\right)_b}^{y}\left[\delta_b^x(V)^{t-1} + \delta_b^y(U)^{t-1}\right]\right\}, \tag{B10}$$

$$F_T = \delta_f^x\left[\overline{A_H}^{x}_{b}\overline{H}^{x}_{b}\delta_b^x(T^{t-1})\right] + \delta_f^y\left[\overline{A_H}^{y}_{b}\overline{H}^{y}_{b}\delta_b^y(T^{t-1})\right], \tag{B11}$$

$$F_S = \delta_f^x\left[(\overline{A_H}^{x}_{b}\overline{H}^{x}_{b}\delta_b^x(S^{t-1})\right] + \delta_f^y\left[\overline{A_H}^{y}_{b}\overline{H}^{y}_{b}\delta_b^y(S^{t-1})\right], \tag{B12}$$

$$F_{q^2} \ = \delta_f^x\left[\overline{\left(\overline{A_M}^{x}_{b}\right)_b}^{z}\overline{H}^{x}_{b}\delta_b^x(q^2)^{t-1}\right] \ + \delta_f^y\left[\overline{\overline{A_M}^{y}_{b}}^{z}_{b}\overline{H}^{y}_{b}\delta_b^y(q^2)^{t-1}\right], \tag{B13}$$

$$F_{q^2 l} = \delta_f^x\left[\overline{\left(\overline{A_M}^{x}_{b}\right)_b}^{z}\overline{H}^{x}_{b}\delta_b^x(q^2 l)^{t-1}\right] + \delta_f^y\left[\overline{\overline{A_M}^{y}_{b}}^{z}_{b}\overline{H}^{y}_{b}\delta_b^y(q^2 l)^{t-1}\right]. \tag{B14}$$

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

$$\frac{\eta^{t+1}-\eta^{t-1}}{2dte} + \delta_f^x(\overline{D}^{x}_{b}\ \ U_A) + \delta_f^y(\overline{D}^{y}_{b}\ \ V_A) = 0, \tag{B15}$$

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

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

$$\delta_f^y\left\{\overline{\left[\overline{(AA_M)_b}^{x}\right]_b}^{y}\overline{\left(\overline{D}^{x}_{b}\right)_b}^{y}\left[\delta_b^x(V_A) + \delta_b^y(U_A)\right]^{t-1}\right\} + \phi_x, \tag{B16}$$

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

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

$$\delta_f^x\left\{\overline{\left[\overline{(AA_M)_b}^{x}\right]_b}^{y}\overline{\left(\overline{D}^{x}_{b}\right)_b}^{y}\left[\delta_b^x(V_A) + \delta_b^y(U_A)\right]^{t-1}\right\} + \phi_y, \tag{B17}$$

where

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

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

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

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

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

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

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

Acknowledgements. 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). Xing Huang is supported by a grant from the State's Key Project of Research and Development Plan (2018YFB0505000). Shixun Zhang is supported by a grant from the State's Key Project of Research and Development Plan (2017YFC1502200) and Qingdao National Laboratory for Marine Science and Technology (QNLM2016ORP0108). Zhenya Song is supported by National Natural Science Foundation of China (U1806205) and AoShan Talents Cultivation Excellent Scholar Program Supported by Qingdao National Laboratory for Marine Science and Technology (2017ASTCP-ES04).

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Griffies, S. M., Böning, C., Bryan, F. O., Chassignet, E. P., Gerdes, R., Hasumi, H., Hirst, A., Treguier, A.-M. and Webb, D.: Developments in ocean climate modelling, Ocean Model., 2(3–4), 123–192, doi:10.1016/S1463-5003(00)00014-7, 2000.

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

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

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

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

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

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

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

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

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

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.

Xu, S., Huang, X., Oey, L. Y., Xu, F., Fu, H., Zhang, Y. and Yang, G.: POM.GPU-v1.0: A GPU-based princeton ocean model, Geosci. Model Dev., doi:10.5194/gmd-8-2815-2015, 2015.

Tables

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

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

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

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**

Table 5. Four benchmark tests

Benchmark	Dimensions	Grid Size	OpenArray	Original
			version (seconds)	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

Figures

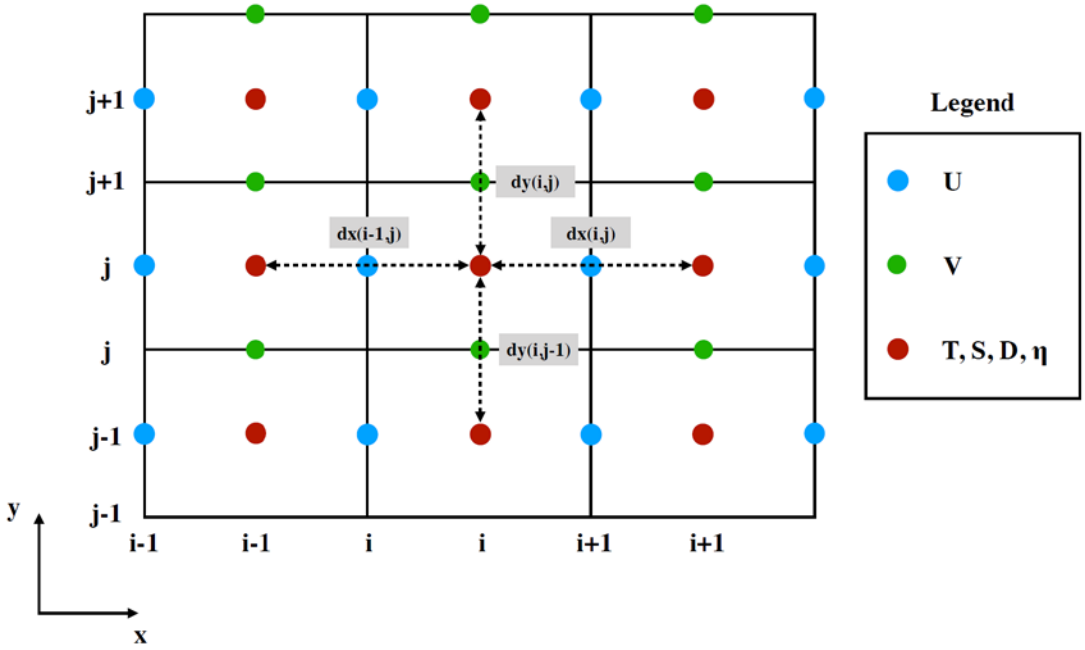
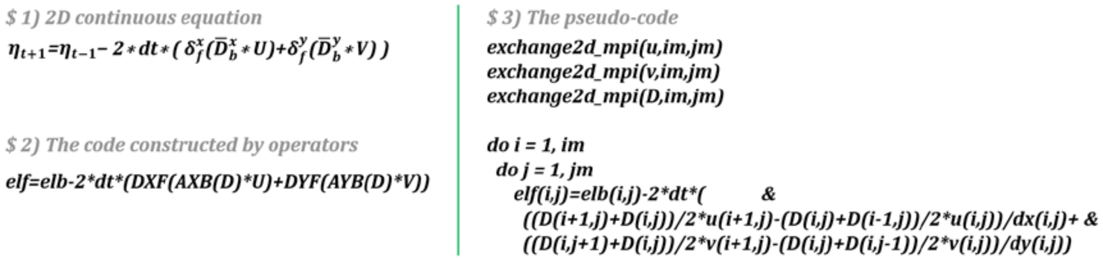


Figure 1. Arrangement of variables in the staggered Arakawa C grid.



$ 1) 2D continuous equation

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

$ 2) The code constructed by operators

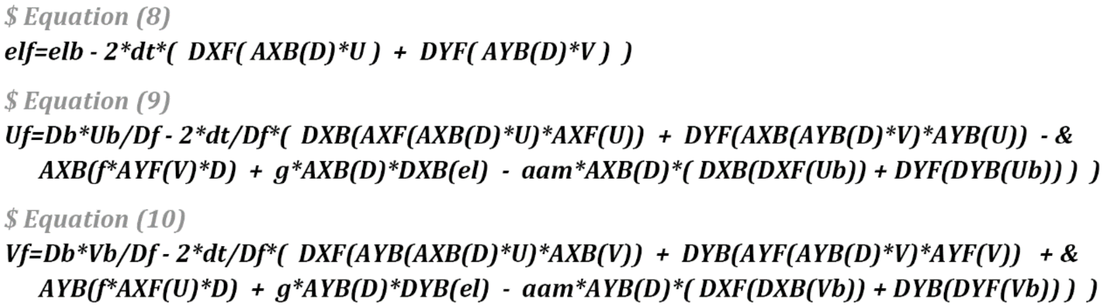
```
elf=elb-2*dt*(DXF(AXB(D)*U)+DYF(AYB(D)*V))
```

$ 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))
```

Figure 2. Implementation of Eq. (6) by basic operators. The *elf* and *elb* are the surface elevations at times (*t+1*) and (*t-1*) respectively.



```
$ 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)) )  )
```

Figure 3. Implementation of the shallow water equations by basic operators. *elf*, *el* and *elb* denote sea surface elevations at times (*t+1*), *t* and (*t-1*), respectively. *Uf*, *U* and *Ub* denote the zonal velocity at times (*t+1*), *t* and (*t-1*), respectively. *Vf*, *V* and *Vb* denote the meridional velocity at times (*t+1*), *t* and (*t-1*), respectively. *aam* denotes the viscosity coefficient.

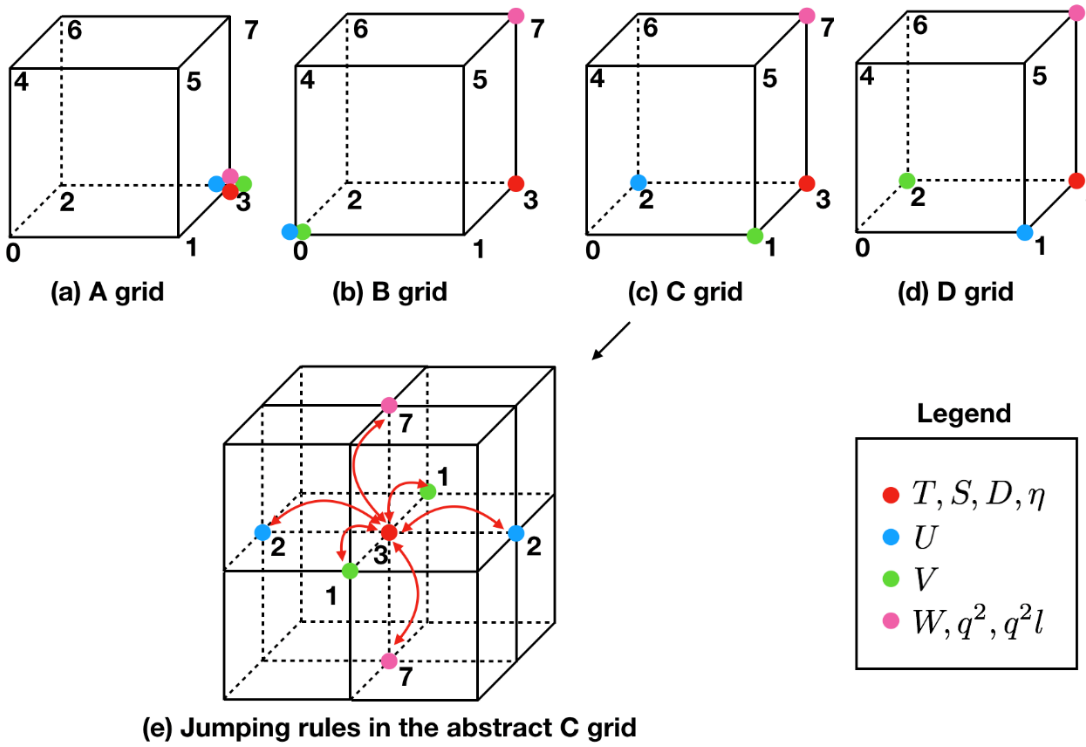


Figure 4. The schematic diagram of the relative positions of the variables on the abstract staggered grid and the jumping procedures among the grid points.

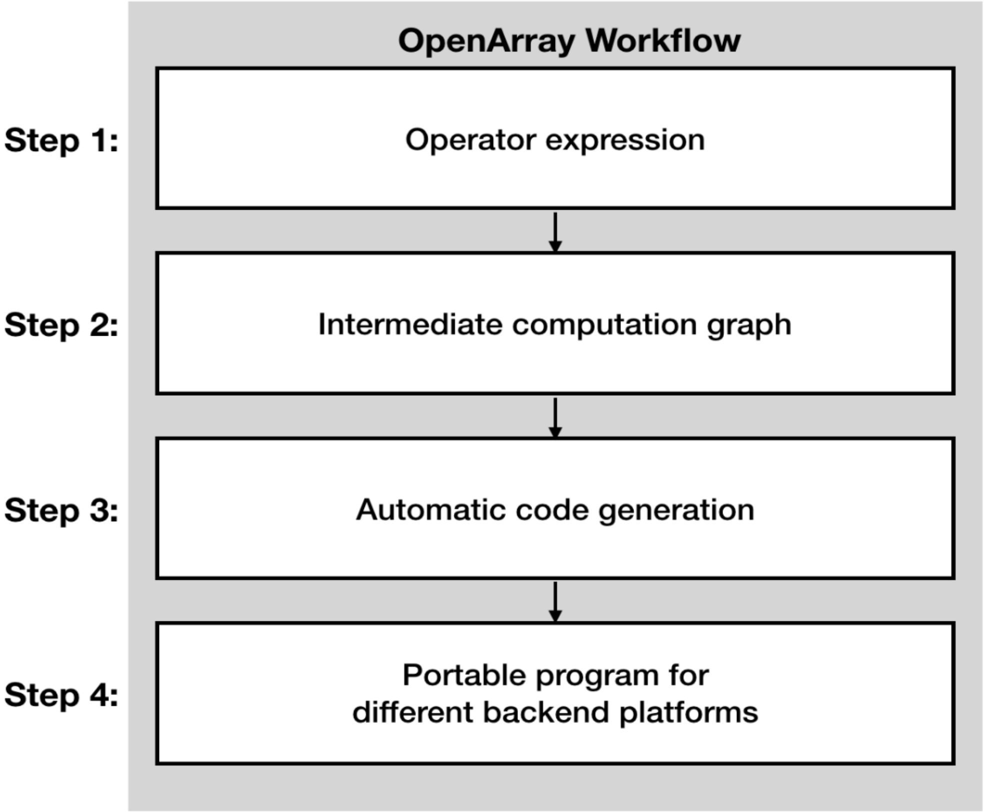


Figure 5. The workflow of OpenArray.

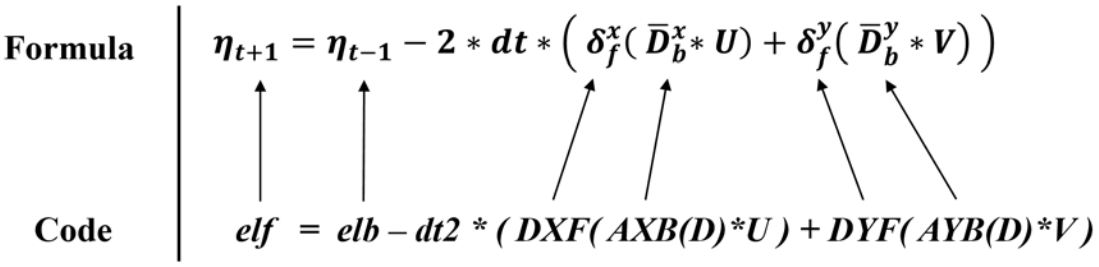
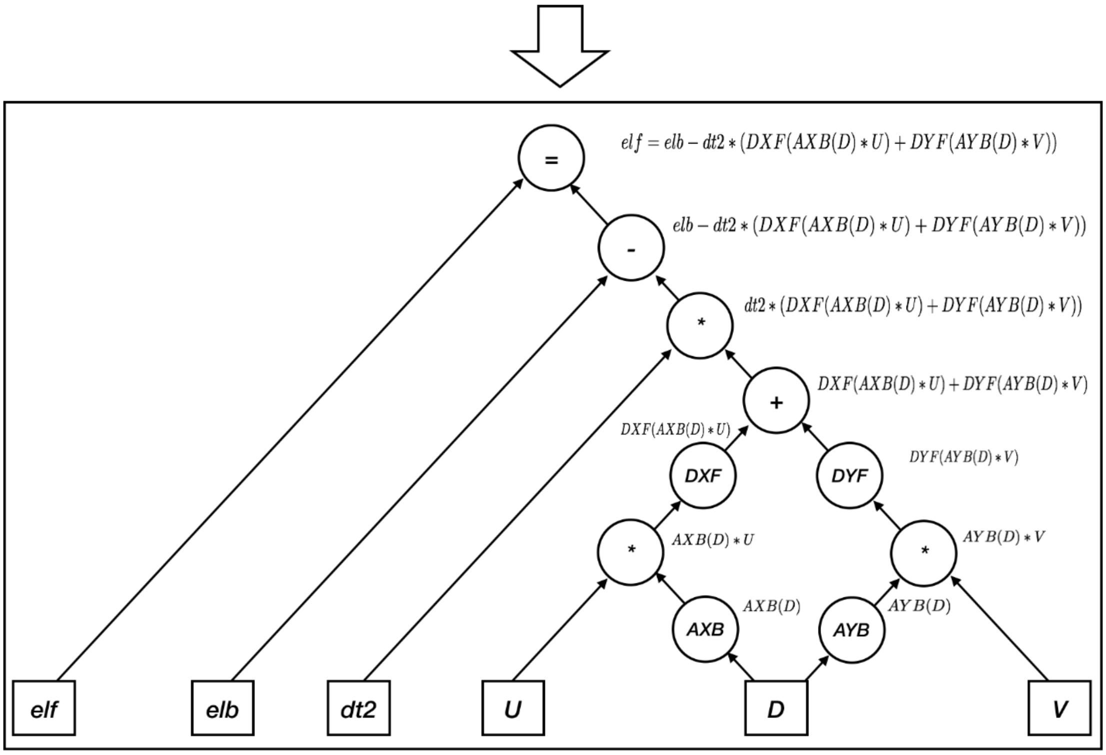

Formula
$\eta_{t+1} = \eta_{t-1} - 2 * dt * \left(\delta_f^x(\bar{D}_b^x * U) + \delta_f^y(\bar{D}_b^y * V) \right)$
Code
*elf = elb – dt2 * (DXF(AXB(D)*U) + DYF(AYB(D)*V)*

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

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



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

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

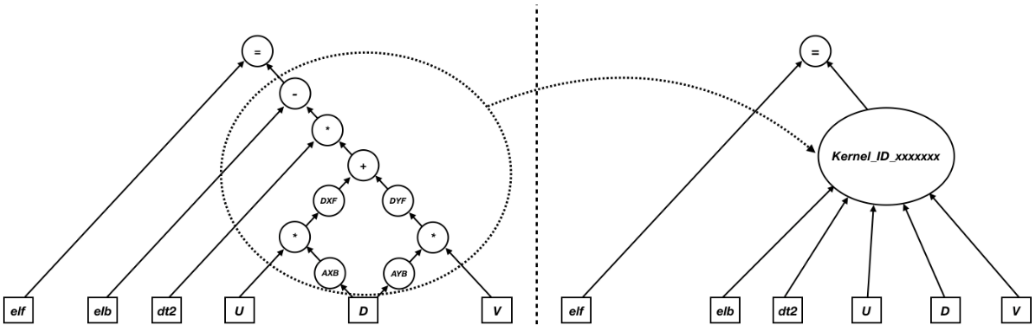


Figure 8. The schematic diagram of kernel fusion.

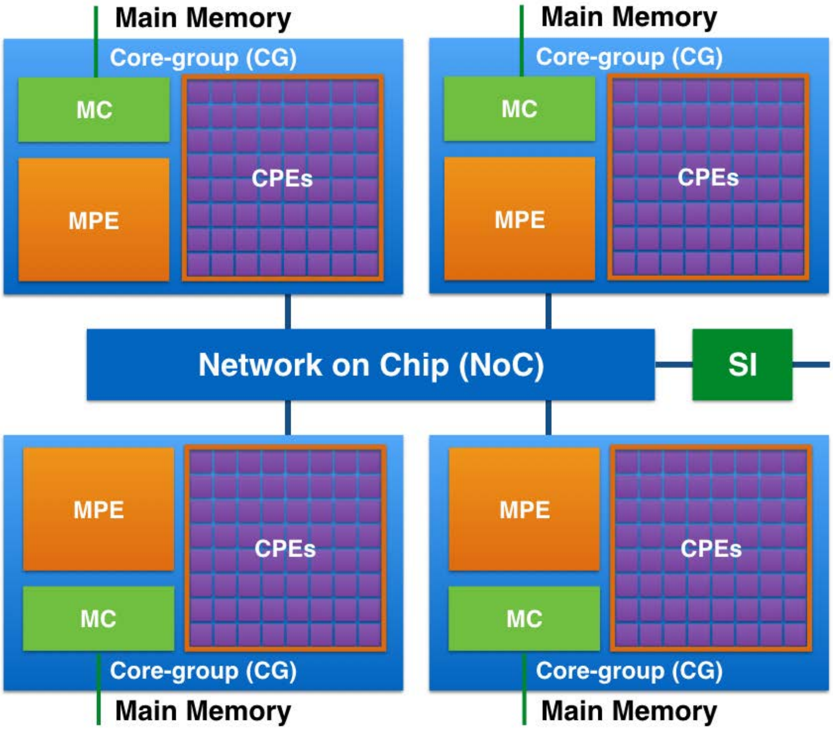


Figure 9. The MPE-CPEs hybrid architecture of the Sunway processor. Every Sunway processor includes 4 Core-groups (CGs) connected by the Network on Chip (NoC). Each CG consists of a management processing element (MPE), 64 computing processing elements (CPEs) and a memory controller (MC). The Sunway processor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

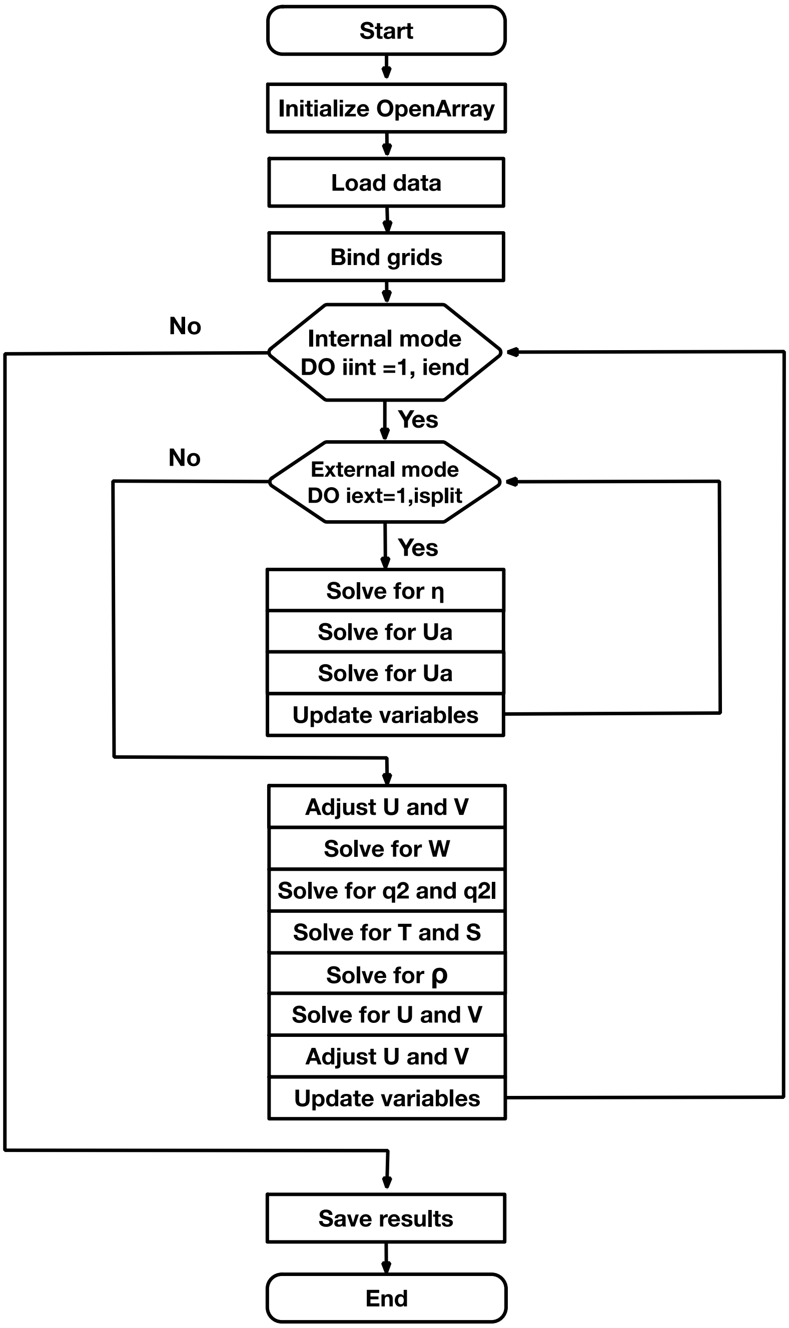


Figure 10. Flow diagram of GOMO

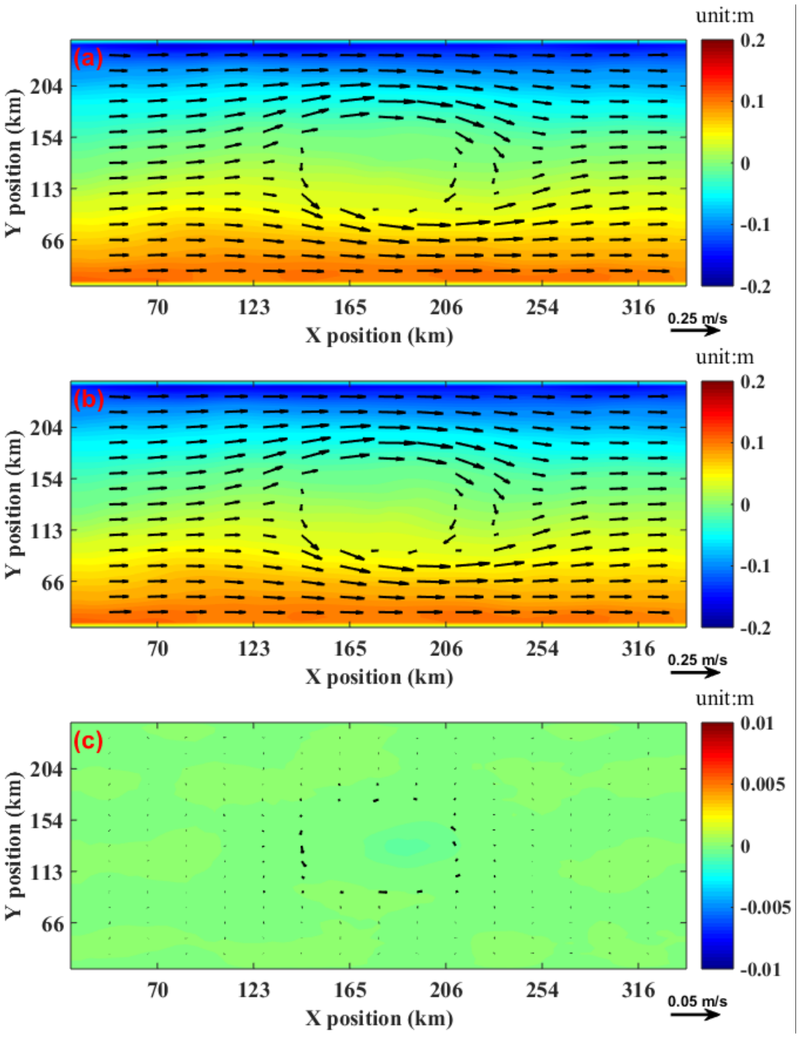


Figure 11. Comparison of the surface elevation (shaded) and currents at 3500 metres depth (vector) between GOMO and sbPOM on the 4th model day. (a) GOMO, (b) sbPOM, (c) GOMO-sbPOM.

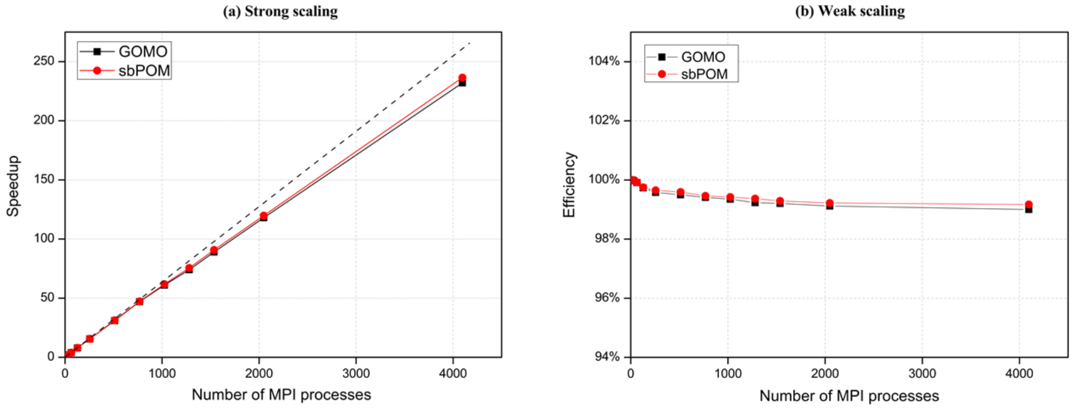


Figure 12. Performance comparison between sbPOM and GOMO on the X86 cluster. (a) The strong scaling result; vertical axis denotes the speedup relative to 16 processes in a single node. (b) The weak scaling result.

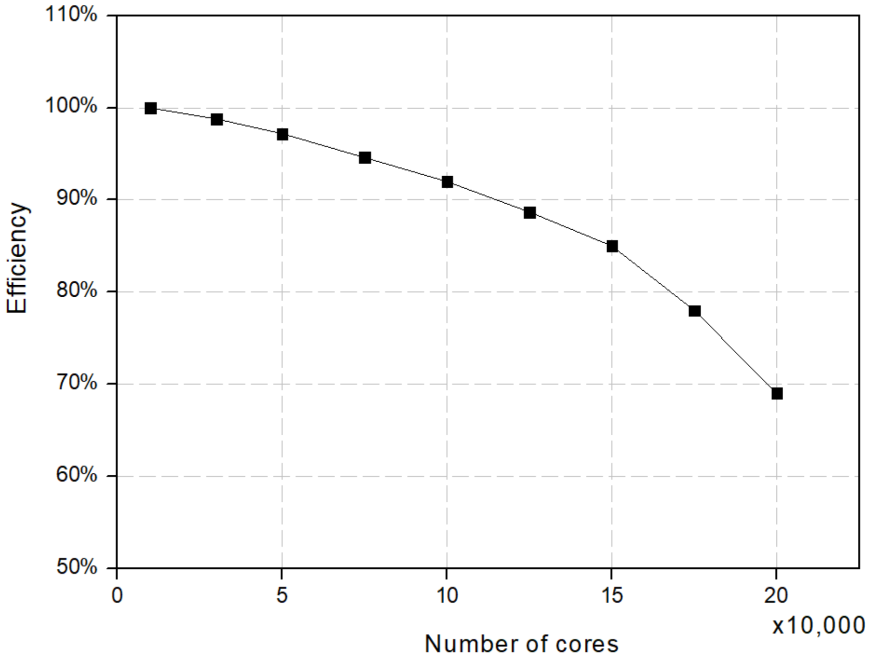


Figure 13. Parallel efficiency of GOMO on the Sunway platform.