Manuscript prepared for Geosci. Model Dev.
with version 2014/09/16 7.15 Copernicus papers of the LaTeX class copernicus.cls.
Date: 27 September 2016

Metos3D: A Marine Ecosystem Toolkit for Optimization and Simulation in 3-D – Part 1: Simulation Package v0.3.2 –

Jaroslaw Piwonski[1] **and Thomas Slawig**[1]

[1]Institute for Computer Science and Kiel Marine Science – Centre for Interdisciplinary Marine Science, Cluster The Future Ocean, Kiel University, 24098 Kiel, Germany. Email: {jpi, ts}@informatik.uni-kiel.de

Correspondence to: Jaroslaw Piwonski (jpi@informatik.uni-kiel.de)

Abstract. We designed and implemented a modular software framework for the off-line simulation of steady cycles of 3-D marine ecosystem models based on the transport matrix approach. It is intended for parameter optimization and model assessment experiments. We defined a software interface for the coupling of a general class of water column-based biogeochemical models, with six models being part of the package. The framework offers both spin-up/fixed-point iteration and Jacobian-free Newton method for the computation of steady states.

The simulation package has been tested with all six models. The Newton method converged for four models when using standard settings, and for two more complex models after alteration of a solver parameter or the initial guess. Both methods delivered the same steady states (within a reasonable precision) on convergence for all models employed, with the Newton iteration generally operating 6 times faster. The effects on performance of both the biogeochemical and the Newton solver parameters were investigated for one model. A profiling analysis was performed for all models used in this work, demonstrating that the number of tracers had a dominant impact on overall performance. We also implemented a geometry-adapted load balancing procedure which showed close to optimal scalability up to a high number of parallel processors.

1 Introduction

In the field of climate research simulations of marine ecosystem models are used to investigate the carbon uptake and storage of earth's oceans. The aim is to identify those processes that play a role in the global carbon cycle. For this purpose coupled simulations of ocean circulation and marine biogeochemistry are required. In this context, marine ecosystems are treated as extensions of biogeochemical systems (cf. Fasham, 2003; Sarmiento and Gruber, 2006). Both terms are therefore used synonymously in this paper. The equations and variables of ocean dynamics are well understood. However, descriptions of biogeochemical or ecological sinks and sources still contain uncertainties with regard to the number of components and to parameterization (cf. Kriest et al., 2010).

To improve this situation a wide range of marine ecosystem models need to be validated, i.e. assessed as to their ability to reproduce real world data. This involves a thorough discussion of simulation results and, before this, an estimation of optimal model parameters for preferably standardized data sets (cf. Fennel et al., 2001; Schartau and Oschlies, 2003).

As a rule hundreds of model evaluations are required for optimization. Therefore any optimization environment for marine ecosystems, which our software framework is intended to supply (as suggested by its name), first and foremost has to provide a fast and flexible simulation framework. In this paper we will concentrate on this prerequisite and present the simulation package of Metos3D. An optimization package will be released subsequently.

For any fully coupled simulation, i.e. simultaneous and interdependent computations of ocean circulation, tracer transport and the biogeochemical sources and sinks in three spatial dimensions, very high computational efforts are needed even at low resolution. Computational complexity increases still more if annual cycles are investigated, since each model evaluation then involves long-time integration (the so-called spin-up) until an equilibrium state is reached under given forcing (cf. Bernsen et al., 2008).

Several strategies have been developed to accelerate computation of periodic steady states in biogeochemical models driven by a 3-D ocean circulation (cf. Bryan, 1984; Danabasoglu et al., 1996; Wang, 2001). We have combined three of

them in our software, namely so-called off-line simulation, optional use of Newton's method for computing steady annual cycles (as an alternative to spin-ups) and spatial parallelization with high scalability.

Off-line simulation affords fundamentally reduced computational costs combined with an acceptable loss of accuracy. The principle is to pre-compute transport data for passive tracers. This approach was adopted by Khatiwala et al. (2005) when introducing the so-called Transport Matrix Method (TMM). The authors used matrices to store the results of a general circulation model, which were then applied to biogeochemical tracer variables. This method proved to be sufficiently accurate to gain first insights into the behavior of biogeochemical models at global basin-scale (cf. Khatiwala, 2007). The software implementation used therein we denote as the *TMM framework* from now on. It is available at Khatiwala (2013).

From the mathematical point of view a steady annual cycle is a periodic solution of a system of (in this case) nonlinear parabolic partial differential equations. This periodic solution is a fixed-point in the mapping that integrates the model variables over one year of model time. Seen in this light a spin-up is a fixed-point iteration. Using an uncomplicated procedure this fixed-point problem can be transformed equivalently into the problem of finding the root(s) of a nonlinear mapping.

Newton-type methods (cf. Dennis and Schnabel, 1996, Chapter 6) are well-known for their superlinear convergence when applied to problems of this kind. When combined with a Krylov subspace approach a Jacobian-free scheme can be realized that is based on evaluations of just one model year (cf. Knoll and Keyes, 2004; Merlis and Khatiwala, 2008; Bernsen et al., 2008).

Whether fixed-point or Newton iteration is used, high performance computing will be needed for running multiple simulations over one year of model time of a 3-D marine ecosystem. Parallel software employing transport matrices and targeting a multi-core distributed-memory architecture requires appropriate data types and linear algebra operations. The specific geometry of oceans with their varying numbers of vertical layers poses an additional challenge for standard load-balancing algorithms – but also offers a chance of developing adapted versions that will improve overall simulation performance. Except for these adaptations our implementation is based on the freely available Portable, Extensible Toolkit for Scientific Computation library (PETSc; Balay et al., 1997, 2012b), which in turn is based on the Message Passing Interface standard (MPI; Walker and Dongarra, 1996).

The objective of this work is to combine three performance-enhancing techniques (off-line computation via transport matrices, Newton method, and highly scalable parallelization) in order to produce a software environment which offers rigorous modularity and complete open-source accessibility. Modularity entails separating data preprocessing and simulation as well as the possibility of implementing any water column-based biogeochemical model with minimal effort. For this purpose we have defined a model interface that permits the use of any number of tracers, parameters, and boundary and domain data. To demonstrate its flexibility we employed an existing biogeochemical model (Dutkiewicz et al., 2005), part of the MITgcm ocean model, as well as a suite of more complex models, which is included in our software package. Our software offers optional use of spin-up/fixed-point iteration or Newton method; for the latter some tuning options were studied. As a result the work of Khatiwala (2008) could be extended by numerically showing convergence for all six models mentioned above without applying preconditioning. Moreover, a detailed profiling analysis of the simulation when using different biogeochemical models demonstrated how the number of tracers impacts overall performance. Finally an adapted load balancing method is presented. It shows scalability that is close to optimal and in this respect is superior to other approaches, including the TMM framework (Khatiwala, 2013).

This paper is structured as follows: In Sections 2 and 3, model equations are described, and the transport matrix approach is recapitulated. In Section 4 both options for computing steady cycles/periodic solutions (fixed-point and Newton iteration) are summarized, and for the latter some tuning options to achieve better convergence are discussed. In Sections 5 and 6, design and implementation of our software package are described, while Section 7 offers a number of numerical results to demonstrate its applicability and performance. Section 8 presents our conclusions, and Section 9 explains how to obtain the source code. The Appendix contains all model equations as well as the parameter settings used for this work; these are available at the same location as the simulation software.

2 Model equations for marine ecosystems

We will consider the following tracer transport model, which is defined by a system of semilinear parabolic partial differential equations (PDEs) of the form

$$\frac{\partial y_i}{\partial t} = \nabla \cdot (\kappa \nabla y_i) - \nabla \cdot (v\, y_i) + q_i(y, u, b, d),\ i = 1, \ldots, n_y, \tag{1}$$

on a time interval $I := [0, T]$ and a spatial domain $\Omega \subset \mathbb{R}^3$ with boundary $\Gamma = \partial\Omega$. $y_i : I \times \Omega \to \mathbb{R}$ denotes a single tracer concentration, and $y = (y_i)_{i=1}^{n_y}$ is the vector of all tracers. Since we are interested in long-time behavior and steady annual cycles, we will assume that the time variable is scaled in years. For brevity's sake we have omitted the dependency on time and space coordinates $(t, \boldsymbol{x})$ in our notation.

The transport of tracers in marine waters is determined by diffusion and advection, which are reflected in the first two linear terms on the right-hand side of (1). Diffusion mix-

ing coefficient $\kappa : I \times \Omega \to \mathbb{R}$ and advection velocity field $v : I \times \Omega \to \mathbb{R}^3$ may either be regarded as given data, or else have to be simulated by an ocean model along with (1). Molecular diffusion of tracers is regarded as negligible compared to turbulent mixing diffusion. Thus κ and both transport terms are the same for all y_i.

Biogeochemical processes within the ecosystem are represented by the last term on the right-hand side of (1), i.e.

$$q_i(y,u,b,d) = q_i(y_1,\dots,y_n,u,b,d), \quad i = 1,\dots,n_y.$$

The functions represented by q_i will often be nonlinear and depend on several tracers, thereby coupling the system. We will refer to the set of functions $q = (q_i)_{i=1}^{n_y}$ as "the biogeochemical model". Typically this model will also depend on parameters. In the software presented in this paper these parameters are assumed to be constant w. r. t. space and time, i.e. we have $u = \boldsymbol{u} \in \mathbb{R}^{n_u}$. For the general setting of (1) this assumption is not necessary. Boundary forcing (e.g. insolation or wind speed, defined on the ocean surface as $\Gamma_s \subset \Gamma$) and domain forcing functions (e.g. salinity or temperature of the ocean water) may also enter into the biogeochemical model. These are denoted by $b = (b_i)_{i=1}^{n_b}, b_i : I \times \Gamma_s \to \mathbb{R}$ and $d = (d_i)_{i=1}^{n_d}, d_i : I \times \Omega \to \mathbb{R}$, respectively.

For tracer transport models, Neumann conditions for the tracers y_i on the boundary Γ are appropriate. They may be either homogeneous (when no tracer fluxes on the boundary are present) or inhomogeneous (to account for flux interactions with atmosphere or sediment, e.g. deposition of nutrients and riverine discharges). In the inhomogeneous case, the necessary data have to be provided as boundary data in b. In Khatiwala (2007, Sect. 3.5) it is shown how the case of tracers with prescribed surface boundary conditions (i.e. Dirichlet conditions) can be treated using the TMM. Then, an appropriate change of the transport matrices is necessary and an additional boundary vector has to be added in every time step.

3 Off-line simulation using transport matrices

The Transport Matrix Method (Khatiwala et al., 2005) allows fast simulation of tracer transport, assuming that forcing data diffusivity κ and advection velocity v are given. This method is based on a discretized counterpart of (1). We introduce the following notation: Let the domain Ω be discretized by a grid $(\boldsymbol{x}_k)_{k=1}^{n_x} \subset \mathbb{R}^3$ and one year in time by $0 = t_0 < \ldots < t_j < t_j + \Delta t_j =: t_{j+1} < \ldots < t_{n_t} = 1$. This means that there are n_t time steps per year. For time instant t_j,

- $\boldsymbol{y}_{ji} = (\boldsymbol{y}_i(t_j, \boldsymbol{x}_k))_{k=1}^{n_x}$ denotes the vector of the values of the i-th tracer at all grid points,
- $\boldsymbol{y}_j = (\boldsymbol{y}_{ji})_{i=1}^{n_y} \in \mathbb{R}^{n_y n_x}$ denotes a vector of the values of *all* tracers at all grid points, appropriately concatenated.

We use analogous notations $\boldsymbol{b}_j, \boldsymbol{d}_j$, and $\boldsymbol{q}_j$ for boundary and domain data and for the biogeochemical terms at the j-th time step. Only corresponding grid points are incorporated for boundary data.

The transport matrix method approximates the discretized counterpart of (1) by

$$\begin{aligned} \boldsymbol{y}_{j+1} &= \mathbf{L}_{imp,j}(\mathbf{L}_{exp,j}\boldsymbol{y}_j + \Delta t_j \boldsymbol{q}_j(\boldsymbol{y}_j, \boldsymbol{u}, \boldsymbol{b}_j, \boldsymbol{d}_j)) \\ &=: \varphi_j(\boldsymbol{y}_j, \boldsymbol{u}, \boldsymbol{b}_j, \boldsymbol{d}_j), \quad j = 0,\dots,n_t - 1. \end{aligned} \tag{2}$$

The linear operators $\mathbf{L}_{exp,j}, \mathbf{L}_{imp,j}$ represent those parts of the transport term in (1) that are discretized explicitly or implicitly w. r. t. time. These operators therefore depend on the given transport data κ, v and thus on time. The biogeochemical term is treated explicitly in (2) using an Euler step.

Since transport affects each tracer individually and is identical for all of them, both $\mathbf{L}_{exp,j}, \mathbf{L}_{imp,j}$ are block-diagonal matrices with n_y identical blocks $\mathbf{A}_{exp,j}, \mathbf{A}_{imp,j} \in \mathbb{R}^{n_x \times n_x}$, respectively. Khatiwala et al. (2005) describes how these matrices can be computed by running one step of an ocean model employing an appropriately chosen set of basis functions for tracer distribution. The operator splitting scheme used in this ocean model therefore determines the partitioning of the transport operator in (1) into an explicit and an implicit matrix. Diffusion (or some part of it) is usually discretized implicitly; in our case this applies only to vertical diffusion. By this procedure we obtain a set of matrix pairs $(\mathbf{A}_{exp,j}, \mathbf{A}_{imp,j})_{j=0}^{n_t-1}$, which usually are sparse. To reduce storing efforts and increase feasibility only a small number of averaged matrices are stored; in our case monthly averages were used. Starting from these matrices, for any time instant t_j an approximation of the matrix pair is computed by linear interpolation.

Thus integration of tracers over one model year only involves sparse matrix-vector multiplications and evaluations of the biogeochemical model. In fact the implicit part of time integration is now pre-computed and contained in $\mathbf{A}_{impl,j}$, which is the benefit of this method. The approximation error of this method when compared to direct coupled computation is determined by the interpolation of transport matrices, the linearization of possibly nonlinear discretization schemes (e.g. flux limiters), and by discounting the reverse influence of ocean biogeochemistry onto circulation fields.

4 Steady annual cycles

The purpose of the software presented in this paper is to allow fast computation of steady annual cycles for the marine ecosystem model under consideration. A steady annual cycle is defined as a periodic solution of (1) with a period length of 1 (year), thus satisfying

$$y(t+1) = y(t), \quad t \in [0,1[.$$

Obviously, the forcing data functions b, d need to be periodic as well.

To apply the transport matrix method we assume that a set of matrices for one model year (generated using this kind of periodic forcing) is available, and that these have been interpolated to obtain pairs $(\mathbf{A}_{exp,j}, \mathbf{A}_{imp,j})$ for all time steps $j = 0, \dots, n_t - 1$. In the discrete setting, a periodic solution will satisfy

$$\boldsymbol{y}_{n_t+j} = \boldsymbol{y}_j \quad j = 0, \dots, n_t - 1.$$

Assuming that the discrete model is completely deterministic, it is sufficient if this equation is satisfied for just one j. In this section we will compare the solutions for the first time instants of two succeeding model years. Defining

$$\boldsymbol{y}^\ell := \boldsymbol{y}_{(\ell-1)n_t} \in \mathbb{R}^{n_y n_x}, \quad \ell = 1, 2, \dots$$

as the vector of tracer values at the first time instant of model year ℓ, a steady annual cycle satisfies

$$\boldsymbol{y}^{\ell+1} = \phi(\boldsymbol{y}^\ell) = \boldsymbol{y}^\ell \text{ in } \mathbb{R}^{n_y n_x} \text{ for some } \ell \in \mathbb{N}, \tag{3}$$

where $\phi := \varphi_{n_t-1} \circ \dots \circ \varphi_0$ is the mapping that performs the tracer integration (2) over one year. All arguments except for $\boldsymbol{y}$ have been omitted in the notation. A steady annual cycle therefore is a fixed-point of the nonlinear mapping ϕ.

Since condition (3) will never be satisfied exactly in a simulation, we measure periodicity, using norms on $\mathbb{R}^{n_y n_x}$ for the residual of (3) . We use the weighted Euclidean norm

$$\|\boldsymbol{z}\|_{2,w} := \left(\sum_{i=1}^{n_y} \sum_{k=1}^{n_x} w_k z_{ik}^2 \right)^{\frac{1}{2}}, w_k > 0, k = 1, \dots, n_x, \tag{4}$$

with $\boldsymbol{z} \in \mathbb{R}^{n_y n_x}$ indexed as $\boldsymbol{z} = ((z_{ik})_{k=1}^{n_x})_{i=1}^{n_y}$. This corresponds to our indexing of tracers, see Section 3. If $w_k = 1$ for all k, we obtain the Euclidean norm denoted by $\|\boldsymbol{z}\|_2$. A stronger correspondence to the continuous problem (1) is achieved by using the discretized counterpart of the $\left(L^2(\Omega)\right)^{n_y}$-norm, where w_k is set to the volume V_k of the k-th grid box. We denote this norm by $\|\boldsymbol{z}\|_{2,V}$. Other settings of weights are possible. All these norms are equivalent in the mathematical sense, i.e. it holds

$$\min_{1 \le k \le n_x} \sqrt{w_k} \, \|\boldsymbol{z}\|_2 \le \|\boldsymbol{z}\|_{2,w} \le \max_{1 \le k \le n_x} \sqrt{w_k} \, \|\boldsymbol{z}\|_2$$

for all $\boldsymbol{z} \in \mathbb{R}^{n_y n_x}$ and all weight vectors $w = (w_k)_{k=1}^{n_x}$ satisfying the positivity condition in Eq. (4).

4.1 Computation by spin-up (fixed-point iteration)

Spin-up signifies repeated application of iteration step (3), in other words, integration in time with fixed forcing until convergence is reached. Based on Banach's fixed-point theorem (cf. Stoer and Bulirsch, 2002) it is well-known that, assuming ϕ is a contractive mapping satisfying

$$\|\phi(\boldsymbol{y}) - \phi(\boldsymbol{z})\| \le L \|\boldsymbol{y} - \boldsymbol{z}\| \quad \text{for all } \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}^{n_y n_x}$$

with $L < 1$ in some norm, this iteration will converge to a unique fixed-point for all initial values $\boldsymbol{y}^0$. This result holds for weaker assumptions as well (cf. Ciric, 1974). This method is quite robust, but shows only linear convergence which is especially slow for $L \approx 1$. An estimation of $L = \max_{\boldsymbol{y}} \|\phi'(\boldsymbol{y})\|$ is difficult, since it involves the Jacobian $\boldsymbol{q}_j'(\boldsymbol{y}_j)$ of the nonlinear biogeochemical model at the current iterate. Typically, thousands of iteration steps (i.e. model years) are needed in order to reach a steady cycle (cf. Bernsen et al., 2008). Moreover, this method offers only restricted options for convergence tuning, the only straightforward one being to choose different time steps Δt_j. For this all transport matrices have to be re-scaled accordingly. The obvious stopping criterion is reduction of the difference between two succeeding iterates measured by

$$\varepsilon_\ell := \|\boldsymbol{y}^\ell - \boldsymbol{y}^{\ell-1}\|_{2,w}$$

in some – optionally weighted – norm.

4.2 Computation by inexact Newton method

By defining $F(\boldsymbol{y}) := \boldsymbol{y} - \phi(\boldsymbol{y})$, the fixed-point problem (3) can be equivalently transformed into the problem of finding a root of $F : \mathbb{R}^{n_y n_x} \to \mathbb{R}^{n_y n_x}$. This problem can be solved by Newton's method (cf. Dennis and Schnabel, 1996; Kelley, 2003; Bernsen et al., 2008). We apply a damped (or globalized) version that incorporates a line search (or backtracking) procedure which (under certain assumptions) provides superlinear and locally even quadratic convergence. Starting from an initial guess $\boldsymbol{y}^0$, in each step the linear system

$$F'(\boldsymbol{y}^m)\boldsymbol{s}^m = -F(\boldsymbol{y}^m) \tag{5}$$

has to be solved, followed by an update $\boldsymbol{y}^{m+1} = \boldsymbol{y}^m + \varrho \boldsymbol{s}^m$. $\varrho > 0$ here denotes the step-size, which is chosen iteratively in such a way that a sufficient reduction in $\|F(\boldsymbol{y}^m + \rho \boldsymbol{s}^m)\|_2$ is achieved (cf. Dennis and Schnabel, 1996, Section 6.3). Note that regarding the Newton solver the Euclidean norm is used. This is determined by the PETSc implementation.

The Jacobian $F'(\boldsymbol{y}^m)$ of F at any current iterate contains the derivative of one model year, thus it is not as sparse as the transport matrices themselves. Therefore a matrix-free version of Newton's method is applied: The linear system (5) is solved by an iterative, so-called Krylov subspace method, which only requires the evaluation of matrix-vector products $F'(\boldsymbol{y}^m)\boldsymbol{s}$. Since $F'(\boldsymbol{y}^m)$ cannot be expected to be symmetric or definite, we use the generalized minimal residual method (GMRES, Saad and Schultz, 1986). The matrix-vector products needed for this can be interpreted as directional derivatives of F at point $\boldsymbol{y}^m$ in the direction of $\boldsymbol{s}$. They may be approximated by a forward finite difference:

$$F'(\boldsymbol{y}^m)\boldsymbol{s} \approx \frac{F(\boldsymbol{y}^m + \delta \boldsymbol{s}) - F(\boldsymbol{y}^m)}{\delta}, \quad \delta > 0. \tag{6}$$

The finite difference step-size δ is chosen automatically as a function of $\boldsymbol{y}^m$ and $\boldsymbol{s}$ (cf. Balay et al., 2012a). An alternative method would be an exact evaluation of the derivative

using the forward mode of Algorithmic Differentiation (cf. Griewank and Walther, 2008).

This approximation of the Jacobian or directional derivative is one reason to call this method *inexact*. The second reason is the fact that the inner linear solver has to be stopped and therefore also is not exact. We use a convergence control procedure based on the technique described by Eisenstat and Walker (1996) for this purpose. Stopping occurs when the Newton residual at the current inner iterate $\boldsymbol{s}$ satisfies

$$\|F'(\boldsymbol{y}^m)\boldsymbol{s} + F(\boldsymbol{y}^m)\|_2 \leq \eta_m \|F(\boldsymbol{y}^m)\|_2. \tag{7}$$

The factor η_m is determined by

$$\eta_m = \gamma \left(\frac{\|F(\boldsymbol{y}^m)\|_2}{\|F(\boldsymbol{y}^{m-1})\|_2} \right)^{\alpha}, \quad m \geq 2, \quad \eta_1 = 0.3. \tag{8}$$

This approach avoids so-called over-solving, i.e. wasting inner steps if the current outer Newton residual $F(\boldsymbol{y}^m)$ is still relatively big. The latter typically occurs at the beginning of Newton iterations. Parameters γ and α can be used to avoid over-solving by adjusting inner accuracy depending on outer accuracy in a linear or nonlinear way, respectively. Moreover, both parameters provide a subtle way to tune the solver. In contrast to a fixed-point iteration, Newton's method even in its damped version may possibly converge only with an appropriately chosen initial guess $\boldsymbol{y}^0$. In a high-dimensional problem such as ours (in $\mathbb{R}^{n_y n_x}$), it is a non-trivial task to find such an initial guess if the standard one used for the spin-up (i.e. a constant tracer distribution) proves unsuccessful. In cases where the Newton iteration proceeds slowly and the criterion described above yields only a few inner iterations, it may be advisable to increase their number by either decreasing γ or increasing α. Below we will give some examples of how convergence may be made possible using this strategy.

In order to estimate the total computational effort needed for the inexact Newton solver and to compare its efficiency with the spin-up method, it must be noted that one evaluation of F basically corresponds to one application of ϕ, i.e. to one model year. Each Newton step requires one evaluation of F as the right-hand side of (5). The initial guess for the inner linear solver iteration is always set at $\boldsymbol{s} = 0$. Thus no computation is required for the first step. For each following inner iteration some evaluation of F is required to compute the second term in the numerator of the right-hand side of (6). The line search may also require additional evaluations of F. Taken together, the overall number of inner iterations plus the overall number of evaluations for the line search determine the number of evaluations of F necessary for this method, which may then be compared to the number of model years needed for the spin-up.

5 Software description

Our software is divided into four repositories, namely `metos3d`, `model`, `data` and `simpack`. The first comprises the installation scripts, the second the biogeochemical model source codes and the third all data preparation scripts as well as the data themselves. The last repository contains the simulation package, i.e. the transport driver, which is implemented in C and based upon the PETSc library. While we have often used 1-indexed arrays within this text for convenience, within the source code C arrays are 0-indexed and Fortran arrays are 1-indexed. All data files are in PETSc format.

5.1 Implementation structure

The implementation of the simulation package is structured in layers as is shown in Figure 21. The layers are organized hierarchically, i.e. each layer provides routines for the layers above. The foundation of the implementation is the PETSc library with its data types and the implementation of the Newton-Krylov solver.

The *bgc* model layer initializes tracer vectors, parameters and boundary and domain data. It is responsible for the interpolation of forcing data and the evaluation of the biogeochemical model (cf. Section 5.3). The *transport* layer is responsible for reading in the transport matrices, interpolating them to the current time step and applying them to the tracer vectors. The main integration routine ϕ (cf. Algorithm 1, 2) is located at the *time stepping* layer. On top resides the *solver* layer, which contains the spin-up implementation and the call to the Newton-Krylov solver.

A call graph for the computation of a steady annual cycle is shown in Figure 22. Note that loops are not explicitly shown therein. Calls to initialization and finalization routines are gathered at the beginning respectively end of a simulation run. The former are responsible for memory allocation and storage of data used at run time. The latter are employed to free memory and delete all vectors and matrices.

The dimensions of the used vectors and matrices depend on the underlying geometry (cf. Section 5.2). The distribution of the work load for a parallel run is determined during initialization of the work load (cf. Section 5.5).

5.2 Geometry information and data alignment

Geometry information is provided as a 2-D land-sea mask plus a designation of the number of vertical layers, i.e. the depth of the different water columns (or *profiles*, cf. Figure 23). This can be understood as a sparse representation of a land-sea cuboid including only wet grid boxes. Hence, the length n_x of a single tracer vector (at fixed time) is the sum of the lengths of all profiles, i.e.

$$n_x = \sum_{k=1}^{n_p} n_{x,k}\,,$$

where n_p is the total number of profiles in the ocean and $(n_{x,k})_{k=1}^{n_p}$ the set of profile lengths. Each profile corresponds to a horizontal gridpoint. Due to the locally varying ocean

depth, the profile lengths depend on the horizontal coordinate, i.e. on the index k.

We denote by $\boldsymbol{y}_{i,k} \in \mathbb{R}^{n_{x,k}}$ the values of the i-th tracer corresponding to the k-th profile at fixed time step. Then the vector of *all* tracers at a fixed time, here denoted by $\boldsymbol{y}$ omitting the time index, can be represented in two ways: Either by *first* collecting all profiles for each tracer and *then* concatenating all tracers, namely

$$\boldsymbol{y} = \left[(\boldsymbol{y}_{1,k})_{k=1}^{n_p} \ \cdots \ (\boldsymbol{y}_{n,k})_{k=1}^{n_p} \right], \tag{9}$$

or vice versa, i.e.

$$\boldsymbol{y} = ((\boldsymbol{y}_{i,k})_{i=1}^{n_y})_{k=1}^{n_p}. \tag{10}$$

In order to multiply matrices with tracer vectors, the first variant is preferable. In order to evaluate a water-column based biogeochemical model, the second one is appropriate.

As a result, all tracers need to be copied from representation (9) to (10) after a transport step. After evaluation of the biogeochemical model we reverse the alignment for the next transport step.

The situation is similar for domain data. Again, we group all domain data profiles by their profile index k, i.e.

$$\left[(\boldsymbol{d}_{1,k})_{k=1}^{n_p} \ \cdots \ (\boldsymbol{d}_{n_d,k})_{k=1}^{n_p} \right] \longrightarrow ((\boldsymbol{d}_{i,k})_{i=1}^{n_d})_{k=1}^{n_p}$$

where $\boldsymbol{d}_{i,k}$ denotes a single domain data profile. However, no reverse copying is required here.

Boundary data have to be treated in a slightly different way. Here we align boundary values, which are associated with the surface of one water column each,

$$\left[(b_{1,k})_{k=1}^{n_p} \ \cdots \ (b_{n_b,k})_{k=1}^{n_p} \right] \longrightarrow ((b_{i,k})_{i=1}^{n_b})_{k=1}^{n_p}$$

where $b_{i,k}$ denotes a single boundary data value as opposed to a whole profile. As with domain data, no reverse copying is required.

5.3 Biogeochemical model interface

One of our main objective in this work is to specify a general coupling interface between the transport induced by ocean circulation and the biogeochemical tracer model. We wish to provide a method to couple any biogeochemical model implementation using any number of tracers, parameters and boundary and domain data to the software that computes the ocean transport. Despite the fact that we consider off-line simulation using transport matrices in this paper only, the interface shall not be restricted to this case. This coupling shall furthermore fit into an optimization context, and it shall be compatible with Algorithmic Differentiation techniques (cf. Section 7).

The only restriction we make for the tracer model is that it operates on each single water column (or *profile*) separately. This means that information on exactly one profile is exchanged via the coupling interface. For models that require information on other profiles (e.g. in the horizontal vicinity) for internal computations, a redefinition of the interface and some internal changes would be necessary. In fact, most of the relevant non-local biogeochemical processes take place within a water column (cf. Evans and Garçon, 1997).

The evaluation of a water-column based biogeochemical model for any fixed time t consists of separate model evaluations for each profile (corresponding to a horizontal spatial coordinate), i.e. for profile index k:

$$\Delta t \, (\boldsymbol{q}_i(t, (\boldsymbol{y}_{i,k})_{i=1}^{n_y}, \boldsymbol{u}, (b_{i,k})_{i=1}^{n_b}, (\boldsymbol{d}_{i,k})_{i=1}^{n_d}))_{i=1}^{n_y}. \tag{11}$$

Here, $(\boldsymbol{y}_{i,k})_{i=1}^{n_y}$ is an input array of n_y tracer profiles according to (10), each with a length or depth of $n_{x,k}$. The vector $\boldsymbol{u}$ contains n_u parameters. Boundary data $(b_{i,k})_{i=1}^{n_b}$ are given as a vector of n_b values, and domain data $(\boldsymbol{d}_{i,k})_{i=1}^{n_d}$ as input array of n_d profiles. Results of the biogeochemical model are stored in the output array $(\boldsymbol{q}_{i,k})_{i=1}^{n_y}$ which also consists of n_y profiles.

Formally speaking this tracer model is scaled from the outside by the (ocean circulation) time step. However, we have integrated Δt into the interface as a concession to the common practice of refining the time step within the tracer model implementation (cf. Kriest et al., 2010). As a consequence, the responsibility for scaling results before returning them to the transport driver software rests with the model implementer.

Listing 1 shows a realization of the biogeochemical model interface in a Fortran 95 subroutine called `metos3dbgc`. The arguments are grouped by data type. The list begins with variables of the type `integer`, i.e. n_y, $n_{x,k}$, n_u, n_b and n_d. These are followed by `real*8` (double precision) arguments, i.e. Δt, $\boldsymbol{q}$, t_j, $\boldsymbol{y}$, $\boldsymbol{u}$, $\boldsymbol{b}$ and $\boldsymbol{d}$. For clarity we have omitted the profile index k and the time index j in our notation. Moreover, we have used `dt` as a textual representation of Δt.

A model initialization and finalization interface is also specified. The former is named `metos3dbgcinit` and the latter `metos3dbgcfinal`. These routines are called at the beginning of each model year, i.e. at t_0, and after the last step of the annual iteration, respectively. Both routines employ the same argument list as `metos3dbgc`. They are not shown here. The names of all three routines are arbitrary and can be altered using pre-processor variables that are defined within `Makefile`.

5.4 Interpolation

Transport matrices as well as boundary and domain data vectors are provided as sets of files. The number of files in each set is arbitrary, although most of the data we use in this work represent a monthly mean.

However, time step counts per model year are generally much higher than the number of available data files. For this reason matrices and vectors are linearly interpolated to the current time step during iteration. All files of a specific data

set are interpreted as averages of the time intervals they represent. We therefore interpolate between the centers of associated intervals. The appropriate weights and indices are computed on the fly using Algorithm 4.

With regard to boundary and domain forcing, we denote data files by $((\boldsymbol{b}_{i,j})_{j=1}^{n_{b,i}})_{i=1}^{n_b}$ and $((\boldsymbol{d}_{i,j})_{j=1}^{n_{d,i}})_{i=1}^{n_d}$. Here, n_b is the number of distinct boundary data sets, and $n_{b,i}$ is the number of data files provided for the i-th set. In the same way, n_d denotes the number of domain data sets and $n_{d,i}$ the number of data files of a particular set.

For every index i and its corresponding boundary data set $(\boldsymbol{b}_{i,j})_{j=1}^{n_{b,i}}$ we compute the appropriate weights α, β as well as indices j_α, j_β and then form a linear combination

$$\boldsymbol{b}_i = \alpha\, \boldsymbol{b}_{i,j_\alpha} + \beta\, \boldsymbol{b}_{i,j_\beta}.$$

The same applies to domain data, i.e. for every domain data set $(\boldsymbol{d}_{i,j})_{j=1}^{n_{d,i}}$ we compute

$$\boldsymbol{d}_i = \alpha\, \boldsymbol{d}_{i,j_\alpha} + \beta\, \boldsymbol{d}_{i,j_\beta}.$$

We use PETSc routines `VecCopy`, `VecScale` and `VecAXPY` for this process.

With regard to transport we have $(\mathbf{A}_{imp,j})_{j=1}^{n_{imp}}$ and $(\mathbf{A}_{exp,j})_{j=1}^{n_{exp}}$ as data files, where n_{imp} and n_{exp} specify the number of implicit and explicit matrix files, respectively. Analogous to the interpolation of vectors we first interpolate all user-provided matrices to the current point in time t_j, i.e. we assemble

$$\mathbf{A} = \alpha\, \mathbf{A}_{j_\alpha} + \beta\, \mathbf{A}_{j_\beta}$$

using the appropriate α, β and j_α, j_β. We use the matrix variants `MatCopy`, `MatScale` and `MatAXPY` for this purpose. The technical details of this process have been discussed in depth in Siewertsen et al. (2013).

To avoid redundant storing we do not assemble both (block diagonal) system matrices during simulation. We use the matrices provided to build just one block for each matrix type instead. The transport step is then applied as a loop over individual tracer vectors.

Unlike vector interpolation and vector operations in general, each matrix operation has a significant impact on computational time. In Section 6.2 we will present results from profiling experiments showing detailed information on the time usage of each operation.

5.5 Load balancing for spatial parallelization

For spatial parallelization, the discrete tracer vectors have to be distributed to the available processes. Since biogeochemical models operate on whole water columns, profiles cannot be split without message passing. But due to the locally varying ocean depth, a tracer vector is a collection of profiles with *different* length. Thus a load balancing that takes into account only the number of profiles, but not their respective length, would be sub-optimal.

The PETSc library provides no load balancing algorithm suitable for this case. We therefore use an approach that was inspired by the idea of space filling curves presented by Zumbusch (1999).

For each profile we compute its 'computational weight', i.e. its mid, in relation to the overall computational effort, i.e. the vector length. We then project this ratio to the available number of processes, i.e we round this figure down to an integer and use the result as the index of the process the profile belongs to. By using this information the profiles can then be assigned consecutively to the processes involved.

For 0-indexed arrays this calculation is described by Algorithm 3. Its theoretical and actual performance is discussed in Section 6.3, where a comparison between Metos3D and the TMM framework is shown.

6 Results

In this section we will present results from our numerical experiments to verify the software. For these experiments the interface described in this paper has been used to couple the transport matrix driver with a suite of biogeochemical models. We will also inspect the convergence behavior of both solvers included in the software. A profiling of the main parts of the algorithm will complement the verification.

In a second step we have performed speed-up tests to analyze the load distribution implemented in our software and compared it with the TMM framework. We will also investigate the convergence control settings of the Newton-Krylov solver and examine the solver's behavior within parameter bounds.

The experimental setup is described in Appendix A in more detail.

6.1 Solver

We begin our verification by computing a steady annual cycle for every model, using both solvers. When using the spin-up we set no tolerance and let the solver iterate for 10,000 model years. The Newton approach is set to a line search variant and the Krylov subspace solver to GMRES. All other settings are left at default, so overall absolute tolerance is at 10^{-8} and the maximum number of inner iterations is 10,000.

The parameter values used for the MITgcm-PO4-DOP model are listed in Table 27 under the heading $\boldsymbol{u}_d$. Table 28 lists the parameter values used for the N, N-DOP, NP-DOP, NPZ-DOP, NPZD-DOP model hierarchy. If not stated otherwise the initial value is set to $2.17\ \mathrm{mmol\,P\,m^{-3}}$ for N or PO4 and $0.0001\ \mathrm{mmol\,P\,m^{-3}}$ for all other tracers.

A comparison of convergence towards a steady annual cycle for both solvers, applied to the MITgcm-PO4-DOP model, is shown in Figure 24. We observe that both solvers reach the same difference between consecutive iterations at the end. Table 29 shows the differences between both solu-

tions in Euclidean and volume-weighted norms, cf. Eq. (4). Figure 25 depicts the difference between both solutions for one tracer at the surface layer. Except for numerical error both solvers obviously compute the same solution.

Figures 26 and 27 show the convergence behavior of both solvers for the N and the N-DOP model, respectively. Again both solvers end with approximately the same accuracy and produce similar results. This impression is confirmed by an inspection of Figures 28 and 29 as well as Table 29.

However, in Figure 210 a different behavior can be observed for the Newton-Krylov solver at the end of the solution process, applied to the NP-DOP model. Closer inspection reveals a peak every 30 model years, which results from the settings of the inner solver, where GMRES is set to perform a restart every 30 years. This option is chosen to reduce the internal storage requirement, but may lead to stagnation for indefinite matrices, cf. Saad (2003, Sect. 6.5.6). It is likely that the Jacobian at some Newton step becomes indefinite, and thus we assume that this is the case here. Figure 211 and Table 29 do not indicate any influence on the solution, however.

For the NPZ-DOP or the NPZD-DOP model the Newton solver shows a different behavior. For both models the solver does not converge if default settings are used, as depicted in Figure 212 (top) and Figure 213 (top). Reduction of the residual per step is quite low, which results in a huge number of iterations. In this case the solver was stopped after 50 iterations (the default setting), which is quite high for Newton's method. This behavior was caused by the fact that convergence of this method – even in its so-called globalized or damped version used here – at times still depends on the initial guess y^0. We therefore used a different one, which was successful with the NPZD-DOP model, see Figure 213 (middle). With the NPZ-DOP model, this procedure still did not work, see Figure 212 (middle).

However, the result of a second and much easier way to achieve convergence can be seen in Figure 212 (top) and Figure 213 (top). If the last Newton iteration step did not lead to a big reduction of the residual, which was obviously the case here, the stopping criterion (8) for the inner iterations of the Newton solver becomes less restrictive. If this criterion is sharpened the number of inner iterations increases and thus the accuracy of the Newton direction improve. This somewhat contradicts the idea formulated in Eisenstat and Walker (1996). Sharpening can easily be achieved by decreasing γ, in this case to $\gamma = 0.3$. This tuning led to convergence, see Figure 212 (bottom) and Figure 213 (bottom). When using these settings the same solutions are obtained as with the spin-up, if numerical errors are neglected (see Figures 214 and 215). This result is confirmed by evaluating the differences in the norm, see Table 29.

It can be observed that as a rule the Newton-Krylov solver does not reach default tolerance within the last Newton step and iterates unnecessarily for 10,000 model years. From now on we will therefore limit the inner Krylov iterations to 200.

For our next investigations using the MITgcm-PO4-DOP model we will alter the convergence settings as well to get rid of the over-solving observed before. More detailed experiments on this subject are presented in Section 6.4.

6.2 Profiling

In the next two sections we will investigate more closely some technical aspects of the implementation. We will first look at the distribution of computational time among the main operations of one model year.

For this purpose we perform a *profiled* sequential run for each model, iterating for 10 model years. An analysis of our profiling results is shown in Figures 216 - 218. When using the MITgcm-PO4-DOP model, for instance, the biogeochemical model takes up 40% of computational time. Interpolation of matrices (`MatCopy`, `MatScale` and `MatAXPY`) amounts to approximately one third. Matrix vector multiplication (`MatMult`) takes up a quarter of all computations and all other operations amount to 0.5%.

Our data also suggest that the greater the number of tracers involved, the more dominant matrix vector multiplication becomes. The `MatMult` operation takes up $19,8\%$ of computational time for the N model, but $56,7\%$ for the NPZD-DOP model. In Table 210 the absolute timings and the computing time per tracer versus number of tracers are shown. The figures confirm the growing dominance of the matrix vector multiplication. The computing time per tracer converges towards $22\,\mathrm{s}$, which is the absolute time spent by the `MatMult` operation per tracer in each model. The absolute timings of the biogeochemical model and the interpolation stay (more or less) constant. They are split among all tracers and thus become less significant. The implications of these results are discussed in Section 7.

Siewertsen et al. (cf. 2013) also made use of this profiling capacity when porting the software to an NVIDIA graphics processing unit (GPU). The authors investigated the impact of the accelerator's hardware on the simulation of biogeochemical models. Their work comprises a detailed discussion of peak performance and memory bandwidth and includes a counting of floating point operations.

6.3 Speed-up

In this section we will investigate in detail the performance of the load balancing algorithm and compare our results with the scalability provided by the TMM framework. We compile both drivers using the same biogeochemical model. We choose the MITgcm-PO4-DOP model using the same time step, initial condition as well as boundary and domain data.

Our tests are run on hardware located at the computing center of Kiel University: an Intel® Sandy Bridge EP architecture with Intel Xeon® E5-2670 CPUs that consist of 16 cores running at 2.6 GHz. We perform 10 tests for our implementation, using 1 to 256 cores.

Each test consists of a simulation run of three model years, where each year is timed separately. For the TMM framework we use 1 to 192 cores and run 5 tests on each core. We use the given output here, which shows the timing for one whole run.

To calculate speed-up and efficiency we use the minimum timings for a specific number of cores. All timings are related to the timing of a sequential run. For a set of computational times $(t_i)_{i=1}^{N}$ measured during our experiments, with $N = 192$ or $N = 256$, respectively, we calculate speed-up as $s_i = t_1/t_i$ and efficiency as $e_i = 100 * s_i/i$.

To investigate the load distribution implemented by us (cf. Section 5.5) we compute the best ratio possible between a sequential and a parallel run. Using Algorithm 3 we first compute the load distribution for all numbers of processes, i.e. $i = 1, \ldots, 260$, and then retrieve the maximum (local) length $n_{i,max}$. To calculate speed-up we divide the vector length by this value, i.e. $s_i = n_y/n_{i,max}$, and to calculate efficiency we again use $e_i = 100 * s_i/i$.

Figure 219 depicts ideal, theoretical and actual data for speed-up and efficiency. Here, the term 'ideal' refers to a perfectly parallelizable program and a perfect hardware with no delay on memory access or communication. Regarding the load distribution implemented by us a good (theoretical) performance can be observed over the whole range of processes. This refers again to a perfect hardware except that we distribute a collection of profiles of different length here.

The data also show that a parallel run of Metos3D on the Intel hardware achieves close to perfect performance when using between 100 and 140 cores. Efficiency is at about 95% in this range and speed-up nearly corresponds to the number of processes. In fact speed-up may rise still further up to slightly over 160, but a minimum of 200 processes are required to achieve this.

In comparison, the scalability of the TMM framework is not optimal. Efficiency drops off immediately and speed-up never rises above 40. For 120 cores and above Metos3D is at least 4 times faster. Interestingly, for low numbers of processes a significant drop in performance can be observed for both drivers. The implications of this are discussed briefly in Section 7. We did not investigate this effect any further, however, since the results presented here already provide a good guideline.

6.4 Convergence control

After this basic verification and the review of some technical aspects of our implementation, we will now investigate those settings that control convergence of the Newton-Krylov solver. Once again we use only the MITgcm-PO4-DOP model. Our intention here is to eliminate the oversolving we observed during the first 200 iterations as shown in Figure 24. This effect occurs if the accuracy of the inner solver is significantly higher than the resulting Newton residual (cf. Eisenstat and Walker, 1996). The relation between these two is controlled by the parameters γ and the α used in Equation (8).

To investigate the influence of these parameters on convergence we compute the reference solution described in Section 6.1 using different values of γ and α. We set overall tolerance to the difference measured between consecutive states after 3,000 model years of spin-up, i.e. approximately 9.0×10^{-4}. γ is varied from 0.5 to 1.0 in steps of 0.1 and α from 1.1 to 1.6, also in steps of 0.1. This makes for a total of 36 model evaluations.

Figure 220 depicts the number of model years and Newton steps required as a function of γ and α. We observe that the overall number of years decreases as the two parameters tend towards 1.0 and 1.1, respectively. In contrast, the number of Newton steps increases, i.e. the Newton residual is computed more often and the inner steps become shorter.

Consequently, since the computation of one residual is negligible in comparison to the simulation of one model year, we focus on decreasing the overall number of model years. A detailed inspection of the results reveals that for $\gamma = 1.0$ and $\alpha = 1.2$ the solver reaches the tolerance set above after approximately 450 model years, which is significantly less than the 600 years needed when using the default settings.

We therefore use these values for our next experiment.

6.5 Parameter samples

So far we have solved the model equations for one (reference) set of parameters only. During optimization, however, solutions must be computed for various parameter sets. Our next experiments therefore investigate the solver's behavior with regard to different model parameters. Once again we only use the MITgcm-PO4-DOP model. Using the MATLAB® routine `lhsdesign`, we create 100 Latin Hypercube (cf. McKay et al., 1979) samples within the bounds described in Table 27. As before we set overall tolerance to a value comparable to 3,000 spin-up iterations and let the Newton solver compute a solution for each parameter sample.

Figure 221 shows histograms of the total number of model years or Newton steps required to solve the model equations. We observe that most computations converge after 400 to 550 model years and require 10 to 30 Newton steps. Interestingly, there is a high peak around 15 and a smaller one around 12 for the Newton method. We also find some outliers in both graphs. Nevertheless all model evaluations we started converged towards a solution within the desired tolerance.

7 Conclusions

We designed and implemented a simulation framework for the computation of steady annual cycles for a generalized class of marine ecosystem models in 3-D, driven by transport matrices pre-computed in an off-line mode. Our framework allows computation of steady cycles by spin-up or by a

globalized Newton method. The software has been realized as open source code throughout.

We also introduced a software interface for water column-based biogeochemical models. We demonstrated the applicability and flexibility of this interface by coupling the biogeochemical component used in the MITgcm general circulation model to the simulation framework. To test the general usability of the interface we then coupled our own implementations of five different biogeochemical models of varying complexity (already used in Kriest et al. (2010)) to the framework. The source code of these models is also available as part of the software package, and may serve as template for the implementation or adaption of other models.

We implemented a transient solver based on the transport matrix approach, where all matrix operations and evaluations of biogeochemical models are performed by spatial parallelization via MPI using the PETSc library. The transport matrices needed for this process are available directly and require no pre-processing.

We realized both a spin-up (or fixed-point iteration) and a globalized Newton solver for the computation of steady cycles. We compared the performance of both solvers and made the following observations: Both delivered the same results (up to a reasonable precision) on convergence. The spin-up converged when using standard sets of parameters, which were taken from Kriest et al. (2010), and equally distributed values for all tracers. The Newton solver did the same for the four models of lower complexity. It did not converge for the other two models when using standard parameter settings and an initial distribution of tracers as described above. For both of these more complex models convergence could be achieved by increasing the number of inner iterations in the Newton solver, which is realized by decreasing the parameter γ in (8). For one of these models convergence could also be achieved by choosing a different initial guess.

With regard to performance, the Newton solver was about 6 times faster for all models. It can be concluded that for complex models the Newton method requires more attention to solver parameter settings, but then is superior to the spin-up, at least when using parameter sets as described above.

In a next step we investigated how performance of the Newton method is influenced by the two solver parameters α, γ in (8), using one model as an example. Employing the optimal choice derived from these experiments (and one model parameter set), we then studied the number of Newton iterations and overall model years needed for 100 latin hypercube model parameter samples. This is an important test for the usability of the Newton method in various kinds of optimization runs, for example if model parameters are varied by the optimizer. As it turned out there was a certain variance in the number of steps needed and thus in the overall effort, but there were no extreme outliers. Our conclusion is that the Newton method is appropriate for optimization, at least for this model, and faster than the usually robust spin-up.

We further analyzed which proportion of computational time is utilized by different parts of our software during simulation of one model year. Our experiments showed that with an increase in the number of tracers the matrix-vector operations started to dominate the process, thus offering the greatest potential for further performance tuning. This was the case even though the transport operator was the same for every tracer. In contrast all biogeochemical interactions contained in the nonlinear coupling terms $\boldsymbol{q}_j$, which mostly are spatially local, become less performance-relevant as the number of tracers increases.

Finally, we implemented a load balancing mechanism which exploits the fact that water columns in the ocean vary in depth, resulting in data vectors of variable length. Using this balancing method a close to optimal speed-up by spatial parallelization was achieved up to the relatively high number of 140 processes. This results in an acceleration factor of four compared to the TMM framework. The factor increases even to five, if 200 processes are used. However, here already 20 % of computational resources are wasted.

To summarize, the software framework presented here offers high flexibility w.r.t. models and steady cycle solvers. The implemented load balancing scheme results in significant improvement in parallel performance. Especially, the apllied Newton solver can be tuned to converge for all six biogeochemical models.

8 Code availability

Name of software: Metos3D (Simulation Package v0.3.2)
Developer: Jaroslaw Piwonski
Year first available: 2012
Software required: PETSc 3.3
Program language: C, C++, Fortran
Size of installation: 1.6 GB
Availability and costs: free software, GPLv3
Software homepage: https://metos3d.github.com/metos3d

The toolkit is maintained using the distributed revision control system `git`. All source codes are available at GitHub (https://github.com). The current versions of `simpack` and `model` are tagged as `v0.3.2`. The data repository is tagged as version `v0.2`. All experiments presented in this work were carried out using these versions. Associated material is stored in the `2016-GMD-Metos3D` repository.

To install the software users should visit the homepage and follow instructions. Future installations will reflect the state of development at that point of time, but users may still retrieve the versions used in this work by invoking `git checkout v0.3.2` in the `simpack` and `model` repository as well as `git checkout v0.2` in the `data` repositories.

Appendix A: Experimental setup

We assume that all PETSc environment variables have been set, the toolkit has been installed and the `metos3d` script has been made available as a shell command.

A1 Models

In order to test our interface we couple an N, N-DOP, NP-DOP, NPZ-DOP, NPZD-DOP model hierarchy as well as an implementation of Dutkiewicz et al. (2005)'s original biogeochemical model. The former has been implemented from scratch for this purpose. The corresponding equations are shown in Appendix B. The latter is the model used for the MIT General Circulation Model (cf. Marshall et al., 1997, MITgcm) biogeochemistry tutorial. We will denote it as the MITgcm-PO4-DOP model.

For every model implementation that is coupled to the transport driver via the interface a new executable must be compiled. We have established naming conventions for the directory structure so that it fits seamlessly into an automatic compile scheme. We create a folder that is named after the biogeochemical model, for instance `MITgcm-PO4-DOP`, within the `model` directory of the `model` repository.

Within this folder the source code file named `model.F` is stored. This directory structure is used for all models. Although the file suffix used here implies a pre-processed Fortran fixed format, any programming language supported by the PETSc library will be accepted.

To compile all sources (still using the same example) we invoke

```
$> metos3d simpack MITgcm-PO4-DOP
```

and obtain an executable named

```
metos3d-simpack-MITgcm-PO4-DOP.exe
```

which we will use for *all* experiments described below. Specific settings will be provided via option files.

A2 Data

All matrices and forcing data used in this work are based on the example material available at (Khatiwala, 2013). This material originates from MITgcm simulations and requires some post-processing. The corresponding preparation scripts are provided along with the processed data in the `data` repository.

The surface grid of the domain used has a longitudinal and latitudinal resolution of $2.8125°$, which produces 128×64 grid points (cf. Figure 23). Note that the Arctic has been filled in, i.e. set to land. This originates in the data provided at the TMM webpage (cf. Khatiwala, 2013). The depth is divided into 15 vertical layers as described in Table 26. This geometry translates to a (single) tracer vector length of $n_x = 52749$ and to $n_p = 4448$ corresponding profiles. Temporal resolution is at $\Delta t = 1/2880$, which is equivalent to an (ocean) time step of 3 hours, assuming that one year consists of 360 days.

The method of computing photosynthetically available short wave radiation is the same for all models. It is deduced from insolation, which is computed on the fly using the formula of Paltridge and Platt (1976). For this purpose latitude and ice cover data are required for the topmost layer, i.e. $n_b = 2$. We use a single latitude file for the former, i.e. $n_{b,1} = 1$, and twelve ice cover files for the latter, $n_{b,2} = 12$.

The depths and heights of all vertical layers are required as well, so we have $n_d = 2$ domain data sets. Each set consists of only one file, i.e. $n_{d,1} = 1$ and $n_{d,2} = 1$. This information is used to compute the attenuation of light by water to determine the fluxes of particulate organic phosphorus and to approximate a derivative with respect to depth. Note that these data sets have to be provided in a specific order, which must correspond to the order used within the model implementation. In addition, twelve implicit transport matrices, i.e. $n_{imp} = 12$, and twelve explicit transport matrices, i.e. $n_{exp} = 12$, are provided as mentioned previously. Each simulation starts at $t_0 = 0$ and performs $n_t = 2880$ iterations per model year.

Appendix B: Model equations

The N, N-DOP, NP-DOP, NPZ-DOP and NPZD-DOP model hierarchy presented here is based on the descriptions used by Kriest et al. (2010). All parameters introduced are shown in Table 28.

B1 Short wave radiation

As mentioned in Section A2, short wave radiation for the *topmost* layer is deduced from insolation, which is computed on the fly using the formula of Paltridge and Platt (1976). For this purpose latitude ϕ and ice cover σ_{ice} data are required. We denote the computed value by $I_{SWR} = I_{SWR}(\phi, \sigma_{ice})$. For all lower layers data on depth $(z_j)_{j=1}^{n_x}$ and height $(dz_j)_{j=1}^{n_x}$ are required. Attenuation by water is described by the coefficient k_w and attenuation by phytoplankton (chlorophyll) by k_c.

B1.1 Implicit phytoplankton

For models N and N-DOP short wave radiation is computed *without* phytoplankton, i.e.

$$I_j = I_{SWR} \begin{cases} I'_j & j = 1 \\ I'_j \prod_{k=1}^{j-1} I_k & \text{else} \end{cases}$$

where $I'_j = \exp(-k_w\, dz_j/2)$, $I_j = \exp(-k_w\, dz_j)$, and j is the index of the individual layers.

B1.2 Explicit phytoplankton

For models NP-DOP, NPZ-DOP and NPZD-DOP short wave radiation is computed *with* phytoplankton included, i.e.

$$I_{P,j} = I_{SWR} \begin{cases} I'_{P,j} & j = 1 \\ I'_{P,j} \prod_{k=1}^{j-1} I_{P,k} & \text{else} \end{cases}$$

where $I'_{P,j} = \exp(-(k_w + k_c\, y_{P,j})\, dz_j/2)$ and $I'_{P,k} = \exp(-(k_w + k_c\, y_{P,k})\, dz_k)$.

B2 N model

The simplest model used here consists of nutrients (N) only, i.e. $\boldsymbol{y} = (\boldsymbol{y}_N)$. The equation is presented in Table B1. Biological uptake is computed as

$$f_P(\boldsymbol{y}_N, I) = \mu_P\, y_P^* \frac{\boldsymbol{y}_N}{K_N + \boldsymbol{y}_N} \frac{I}{K_I + I},$$

where the implicitly prescribed concentration of phytoplankton is set to $y_P^* = 0.0028\ \mathrm{mmol\,P\,m^{-3}}$. Note that y_P^* could be a free model parameter as well. However, we stick to this formulation to be consistent with Kriest et al. (2010). The N model introduces $n_u = 5$ parameters, with $\boldsymbol{u} = (k_w, \mu_P, K_N, K_I, b)$.

B3 N-DOP model

The N-DOP model consists of nutrients (N) and dissolved organic phosphorus (DOP), i.e. $\boldsymbol{y} = (\boldsymbol{y}_N, \boldsymbol{y}_{DOP})$. Computation of biological uptake remains the same. The equations are shown in Table B2. The N-DOP model introduces $n_u = 7$ parameters, with $\boldsymbol{u} = (k_w, \mu_P, K_N, K_I, \sigma_{DOP}, \lambda_{DOP}, b)$.

B4 NP-DOP model

The NP-DOP model consists of nutrients (N), phytoplankton (P), and dissolved organic phosphorus (DOP), i.e. $\boldsymbol{y} = (\boldsymbol{y}_N, \boldsymbol{y}_P, \boldsymbol{y}_{DOP})$. Here nutrient uptake by (explicit) phytoplankton is computed as

$$f_P(\boldsymbol{y}_N, \boldsymbol{y}_P, I_P) = \mu_P\, \boldsymbol{y}_P \frac{\boldsymbol{y}_N}{K_N + \boldsymbol{y}_N} \frac{I_P}{K_I + I_P}.$$

Computation of short wave radiation is altered as well (see Section B1.2). In addition a quadratic loss term for phytoplankton is introduced, as is a grazing function

$$f_Z(\boldsymbol{y}_P) = \mu_Z\, y_Z^* \frac{\boldsymbol{y}_P^2}{K_P^2 + \boldsymbol{y}_P^2},$$

where the implicitly prescribed concentration of zooplankton is set to $y_Z^* = 0.01\ \mathrm{mmol\,P\,m^{-3}}$. Again, we stick to this formulation to be consistent with Kriest et al. (2010), though y_Z^* could be a free model parameter. The equations are shown in Table B3. The NP-DOP model introduces $n_u = 13$ parameters, with $\boldsymbol{u} = (k_w, k_c, \mu_P, \mu_Z, K_N, K_P, K_I, \sigma_{DOP}, \lambda_P, \kappa_P, \lambda'_P, \lambda_{DOP}, b)$.

B5 NPZ-DOP model

The NPZ-DOP model consists of nutrients (N), phytoplankton (P) zooplankton (Z) and dissolved organic phosphorus (DOP), i.e. $\boldsymbol{y} = (\boldsymbol{y}_N, \boldsymbol{y}_P, \boldsymbol{y}_Z, \boldsymbol{y}_{DOP})$. The production function remains the same. For the computation of grazing, zooplankton is dealt with explicitly, i.e.

$$f_Z(\boldsymbol{y}_P, \boldsymbol{y}_Z) = \mu_P\, \boldsymbol{y}_Z \frac{\boldsymbol{y}_P^2}{K_P^2 + \boldsymbol{y}_P^2}.$$

The equations are shown in Table B4. The NPZ-DOP model introduces $n_u = 16$ parameters, with $\boldsymbol{u} = (k_w, k_c, \mu_P, \mu_Z, K_N, K_P, K_I, \sigma_Z, \sigma_{DOP}, \lambda_P, \lambda_Z, \kappa_Z, \lambda'_P, \lambda'_Z, \lambda'_{DOP}, b)$.

B6 NPZD-DOP model

The NPZD-DOP model consists of nutrients (N), phytoplankton (P) zooplankton (Z), detritus (D) and dissolved organic phosphorus (DOP), i.e. $\boldsymbol{y} = (\boldsymbol{y}_N, \boldsymbol{y}_P, \boldsymbol{y}_Z, \boldsymbol{y}_D, \boldsymbol{y}_{DOP})$. Most equations are unchanged, except that a depth-dependent linear sinking speed is introduced for detritus. The equations are shown in Table B5. The NPZD-DOP model introduces $n_u = 16$ parameters, with $\boldsymbol{u} = (k_w, k_c, \mu_P, \mu_Z, K_N, K_P, K_I, \sigma_Z, \sigma_{DOP}, \lambda_P, \lambda_Z, \kappa_Z, \lambda'_P, \lambda'_Z, \lambda'_D, \lambda'_{DOP}, a_D, b_D)$.

Acknowledgements. The authors would like to thank S. Khatiwala for providing support on the subject of transport matrices and for offering the whole TMM material freely on the internet. Furthermore, both authors would like to thank I. Kriest and A. Oschlies for many fruitful discussions. In particular, Jaroslaw Piwonski would like to thank I. Kriest for teaching him patiently so much about biogeochemical models. This work was partly funded by the DFG cluster *The Future Ocean*, grant EXC 80 and by the German Federal Ministry of Education and Research (BMBF), grant 01LP1512B. The sole responsibility for the report's content lies with the authors. Finally the authors would like to thank the editor, an anonymous referee and Momme Butenschön for their valuable comments.

References

Balay, S., Gropp, W. D., McInnes, L. C., and Smith, B. F.: Efficient Management of Parallelism in Object Oriented Numerical Software Libraries, in: Modern Software Tools in Scientific Computing, edited by Arge, E., Bruaset, A. M., and Langtangen, H. P., pp. 163–202, Birkhäuser Press, Basel, 1997.

Balay, S., Brown, J., , Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H.: PETSc Users Manual, Tech. Rep. ANL-95/11 - Revision 3.3, Argonne National Laboratory, Lemont, 2012a.

Balay, S., Buschelman, K., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H.: PETSc Web page, http://www.mcs.anl.gov/petsc/ (last access: 12 July 2013), 2012b.

Table B1. Equations for the N model with $f_P = f_P(y_N, I)$ and $E_j = f_P\, dz_j$.

	Euphotic zone	Sinking
$q_N(y) =$	$-f_P$	$+E_j\, \partial_z (z/z_j)^{-b}$

Table B2. Equations for the N-DOP model with $f_P = f_P(y_N, I)$ and $E_j = \bar{\sigma}_{DOP}\, f_P\, dz_j$.

	Euphotic zone	All layers	Sinking
$q_N(y) =$	$-f_P$	$+\lambda'_{DOP}\, y_{DOP}$	$+E_j\, \partial_z (z/z_j)^{-b}$
$q_{DOP}(y) =$	$+\sigma_{DOP} f_P$	$-\lambda'_{DOP}\, y_{DOP}$	

Table B3. Equations for the NP-DOP model with $f_P = f_P(y_N, y_P, I_P)$, $f_Z = f_Z(y_P)$ and $E_j = \bar{\sigma}_{DOP}\, f_Z\, dz_j$.

	Euphotic zone				All layers		Sinking
$q_N(y) =$	$-f_P$					$+\lambda'_{DOP}\, y_{DOP}$	$+E_j\, \partial_z (z/z_j)^{-b}$
$q_P(y) =$	$+f_P$	$-f_Z$	$-\lambda_P\, y_P$	$-\kappa_P\, y_P^2$	$-\lambda'_P\, y_P$		
$q_{DOP}(y) =$		$+\sigma_{DOP}\, f_Z$	$+\lambda_P\, y_P$	$+\kappa_P\, y_P^2$	$+\lambda'_P\, y_P$	$-\lambda'_{DOP}\, y_{DOP}$	

Table B4. Equations for the NPZ-DOP model with $f_P = f_P(y_N, y_P, I_P)$, $f_Z = f_Z(y_P, y_Z)$ and $E_j = \bar{\sigma}_{DOP}(\bar{\sigma}_Z\, f_Z + \lambda_P\, y_P + \kappa_Z\, y_Z^2)\, dz_j$.

	Euphotic zone					All layers			Sinking
$q_N(y) =$	$-f_P$			$+\lambda_Z\, y_Z$				$+\lambda'_{DOP}\, y_{DOP}$	$+E_j\, \partial_z (z/z_j)^{-b}$
$q_P(y) =$	$+f_P$	$-f_Z$	$-\lambda_P\, y_P$			$-\lambda'_P\, y_P$			
$q_Z(y) =$		$+\sigma_Z\, f_Z$		$-\lambda_Z\, y_Z$	$-\kappa_Z\, y_Z^2$		$-\lambda'_Z\, y_Z$		
$q_{DOP}(y) =$		$+\sigma_{DOP}(\bar{\sigma}_Z\, f_Z$	$+\lambda_P\, y_P$		$+\kappa_Z\, y_Z^2)$	$+\lambda'_P\, y_P$	$+\lambda'_Z\, y_Z$	$-\lambda'_{DOP}\, y_{DOP}$	

Table B5. Equations for the NPZD-DOP model with $f_P = f_P(y_N, y_P, I_P)$ and $f_Z = f_Z(y_P, y_Z)$.

	Euphotic zone					All layers				Sinking
$q_N(y) =$	$-f_P$				$+\lambda_Z\, y_Z$			$+\lambda'_D\, y_D$	$+\lambda'_{DOP}\, y_{DOP}$	
$q_P(y) =$	$+f_P$	$-f_Z$	$-\lambda_P\, y_P$			$-\lambda'_P\, y_P$				
$q_Z(y) =$		$+\sigma_Z\, f_Z$		$-\kappa_Z\, y_Z^2$	$-\lambda_Z\, y_Z$		$-\lambda'_Z\, y_Z$			
$q_D(y) =$		$+\bar{\sigma}_{DOP}\,(\bar{\sigma}_Z\, f_Z$	$+\lambda_P\, y_P$	$+\kappa_Z\, y_Z^2)$				$-\lambda'_D\, y_D$		$+\partial_z w(z)\, y_D$
$q_{DOP}(y) =$		$+\sigma_{DOP}\,(\bar{\sigma}_Z\, f_Z$	$+\lambda_P\, y_P$	$+\kappa_Z\, y_Z^2)$		$+\lambda'_P\, y_P$	$+\lambda'_Z\, y_Z$		$-\lambda'_{DOP}\, y_{DOP}$	

Bernsen, E., Dijkstra, H. A., and Wubs, F. W.: A method to reduce the spin-up time of ocean models, Ocean Modelling, 20, 380 – 392, doi:10.1016/j.ocemod.2007.10.008, 2008.

Bryan, K.: Accelerating the Convergence to Equilibrium of Ocean-Climate Models, Journal of Physical Oceanography, 14, 666–673, doi:10.1175/1520-0485(1984)014<0666:ATCTEO>2.0.CO;2, 1984.

Ciric, L. B.: A Generalization of Banach's Contraction Principle, Proceedings of the American Mathematical Society, 45, 267–273, 1974.

Danabasoglu, G., McWilliams, J. C., and Large, W. G.: Approach to Equilibrium in Accelerated Global Oceanic Models, Journal of Climate, 9, 1092–1110, doi:10.1175/1520-0442(1996)009<1092:ATEIAG>2.0.CO;2, 1996.

Dennis, J. and Schnabel, R.: Numerical methods for unconstrained optimization and nonlinear equations, Society for Industrial and Applied Mathematics, Philadelphia, 1996.

Dutkiewicz, S., Sokolov, A. P., Scott, J., and Stone, P. H.: A three-dimensional ocean-seaice-carbon cycle model and its coupling to a two-dimensional atmospheric model: Uses in climate change studies, Tech. Rep. 122, MIT Joint Program on the Science and Policy of Global Change, Cambridge, 2005.

Eisenstat, S. C. and Walker, H. F.: Choosing the Forcing Terms in an Inexact Newton Method, SIAM Journal on Scientific Computing, 17, 16–32, doi:10.1137/0917003, 1996.

Evans, G. T. and Garçon, V. C.: One-Dimensional Models of Water Column Biogeochemistry, Report of a workshop held in Toulouse, France, November-December 1995. GOFS Report N°23/97, JGOFS Bergen, Norway, 1997.

Fasham, M. J. R., ed.: Ocean Biogeochemistry. The Role of the Ocean Carbon Cycle in Global Change., Global Change – The IGBP Series, Springer, Berlin et al., 2003.

Fennel, K., Losch, M., Schröter, J., and Wenzel, M.: Testing a marine ecosystem model: sensitivity analysis and parameter optimization, Journal of Marine Systems, 28, 45–63, doi:10.1016/S0924-7963(00)00083-X, 2001.

Griewank, A. and Walther, A.: Evaluating derivatives: principles and techniques of algorithmic differentiation, Society for Industrial and Applied Mathematics (SIAM), 2008.

Kelley, C. T.: Solving nonlinear equations with Newton's method, SIAM, Philadelphia, 2003.

Khatiwala, S.: A computational framework for simulation of biogeochemical tracers in the ocean, Global Biogeochemical Cycles, 21, doi:10.1029/2007GB002923, 2007.

Khatiwala, S.: Fast spin up of Ocean biogeochemical models using matrix-free Newton-Krylov, Ocean Modelling, 23, 121–129, doi:10.1016/j.ocemod.2008.05.002, 2008.

Khatiwala, S.: TMM framework web page, http://www.ldeo.columbia.edu/%7Espk/Research/TMM/ (last access: 14 June 2016), 2013.

Khatiwala, S., Visbeck, M., and Cane, M.: Accelerated simulation of passive tracers in ocean circulation models, Ocean Modelling, 9, 51–69, 2005.

Knoll, D. and Keyes, D.: Jacobian-free Newton–Krylov methods: a survey of approaches and applications, Journal of Computational Physics, 193, 357–397, 2004.

Kriest, I., Khatiwala, S., and Oschlies, A.: Towards an assessment of simple global marine biogeochemical models of different complexity, Progress In Oceanography, 86, 337–360, doi:10.1016/j.pocean.2010.05.002, 2010.

Marshall, J., Adcroft, A., Hill, C., Perelman, L., and Heisey, C.: A finite-volume, incompressible Navier Stokes model for studies of the ocean on parallel computers, Journal of Geophysical Research, 102, 5753–5766, 1997.

McKay, M. D., Beckman, R. J., and Conover, W. J.: Comparison of three methods for selecting values of input variables in the analysis of output from a computer code, Technometrics, 21, 239–245, 1979.

Merlis, T. M. and Khatiwala, S.: Fast dynamical spin-up of ocean general circulation models using Newton–Krylov methods, Ocean Modelling, 21, 97–105, 2008.

Paltridge, G. W. and Platt, C. M. R.: Radiative Processes in Meteorology and Climatology, Elsevier, New York, doi:10.1002/qj.49710343713, 1976.

Saad, Y.: Iterative Methods for Sparse Linear Systems, The Society for Industrial and Applied Mathematics (SIAM), 2003.

Saad, Y. and Schultz, M.: GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, SIAM Journal on Scientific and Statistical Computing, 7, 856–869, doi:10.1137/0907058, 1986.

Sarmiento, J. L. and Gruber, N.: Ocean Biogeochemical Dynamics, Princeton University Press, Princeton et al., 2006.

Schartau, M. and Oschlies, A.: Simultaneous data-based optimization of a 1d-ecosystem model at three locations in the north Atlantic: Part I - method and parameter estimates, Journal of Marine Research 61, pp. 765–793, 2003.

Siewertsen, E., Piwonski, J., and Slawig, T.: Porting marine ecosystem model spin-up using transport matrices to GPUs, Geoscientific Model Development, 6, 17–28, doi:10.5194/gmd-6-17-2013, 2013.

Stoer, J. and Bulirsch, R.: Introduction to Numerical Analysis, Springer, New York, 3rd edn., 2002.

Walker, D. W. and Dongarra, J. J.: MPI: A Standard Message Passing Interface, Supercomputer, 12, 56–68, 1996.

Wang, D.: A note on using the accelerated convergence method in climate models, Tellus A, 53, 27–34, doi:10.1034/j.1600-0870.2001.01134.x, 2001.

Zumbusch, G. W.: Dynamic Load Balancing in a Lightweight Adaptive Parallel Multigrid PDE Solver., in: PPSC, SIAM, Philadelphia, 1999.

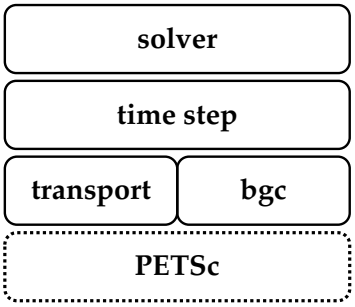

Figure 21. Implementation layers of the Metos3D simulation package (cf. Section 5.1).

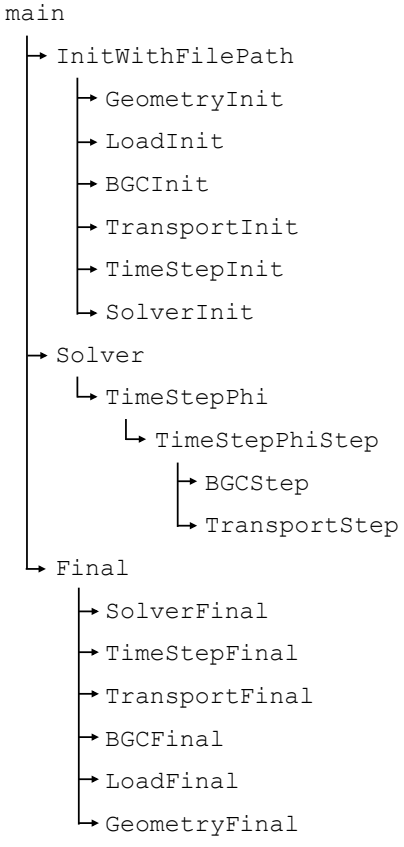

Figure 22. Call graph for the computation of a steady annual cycle(cf. Section 5.1).

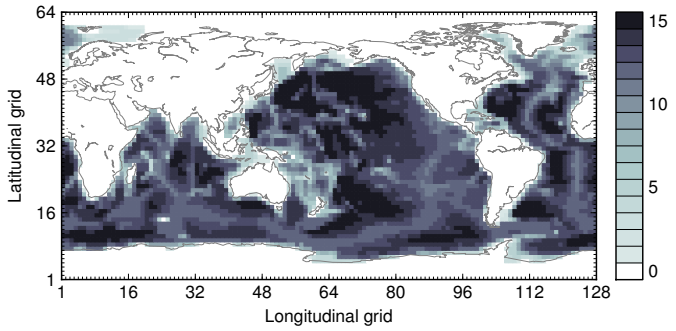

Figure 23. Land-sea mask (geometric data) of the used numerical model. Shown are the number of layers per grip point. Note that the Arctic has been filled in.

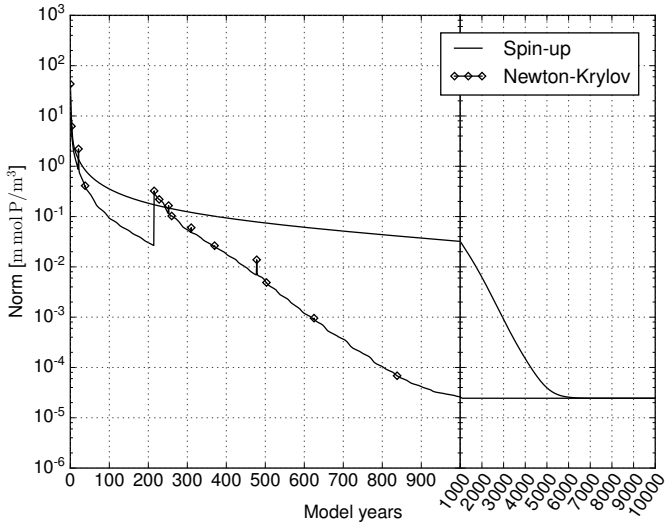

Figure 24. *MITgcm-PO4-DOP model:* Convergence towards an annual cycle. *Spin-up:* norm of difference between initial states of consecutive model years (solid line). *Newton-Krylov:* residual norm at a Newton step (diamond) and norm of the GMRES residual during solving (solid line in-between).

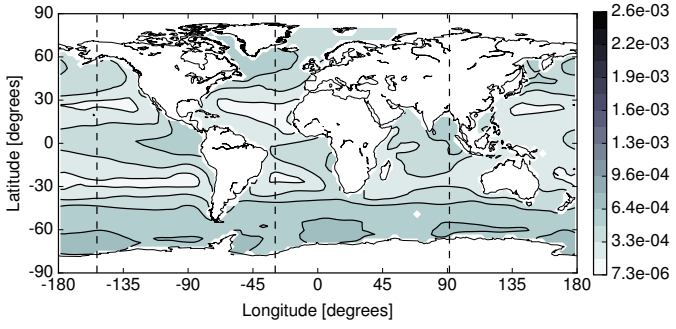

Figure 25. *MITgcm-PO4-DOP model:* Difference between the phosphate concentration of the spin-up and the Newton solution at the first layer (0 – 50 m) in the Euclidean norm. Units are $\mathrm{mmol\,P\,m^{-3}}$.

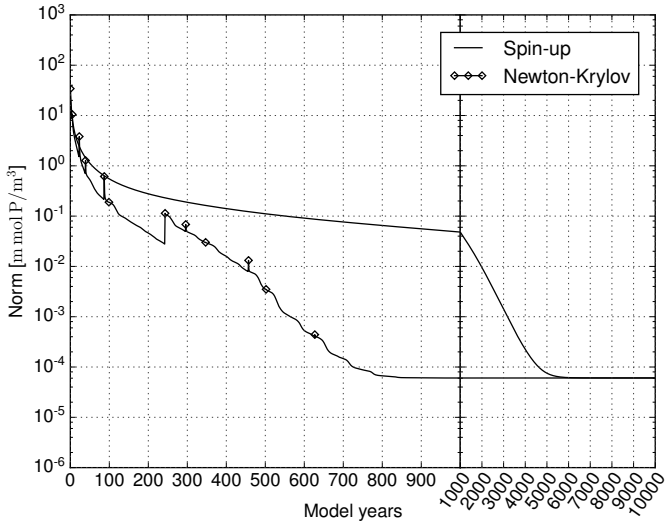

Figure 26. *N model:* Convergence towards an annual cycle using spin-up and Newton-Krylov solver.

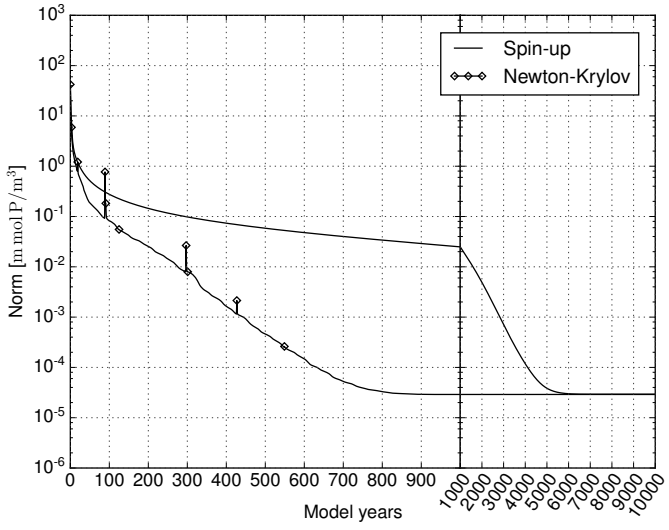

Figure 27. *N-DOP model:* Convergence towards an annual cycle using a spin-up and a Newton-Krylov solver.

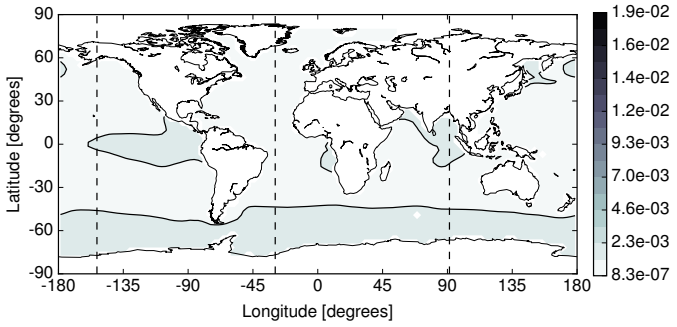

Figure 28. *N model:* Difference between the phosphate concentration of the spin-up and the Newton solution at the first layer (0 – 50 m) in the Euclidean norm. Units are $\mathrm{mmol\,P\,m^{-3}}$.

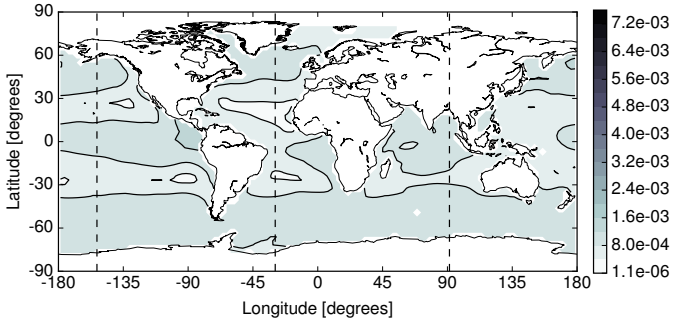

Figure 29. *N-DOP model:* Difference between the phosphate concentration of the spin-up and the Newton solution at the first layer (0 – 50 m) in the Euclidean norm. Units are $\mathrm{mmol\,P\,m^{-3}}$.

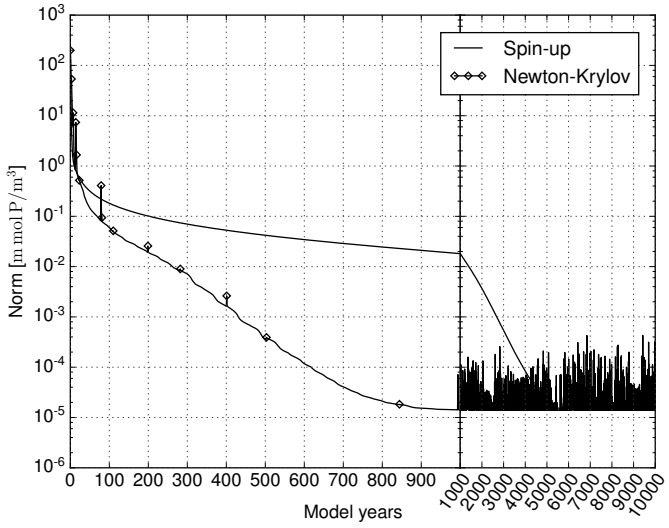

Figure 210. *NP-DOP model:* Convergence towards an annual cycle using a spin-up and a Newton-Krylov solver.

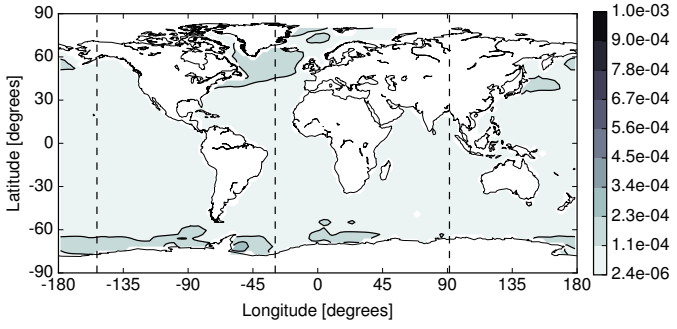

Figure 211. *NP-DOP model:* Difference between the phosphate concentration of the spin-up and the Newton solution at the first layer (0 – 50 m) in the Euclidean norm. Units are $\mathrm{mmol\,P\,m^{-3}}$.

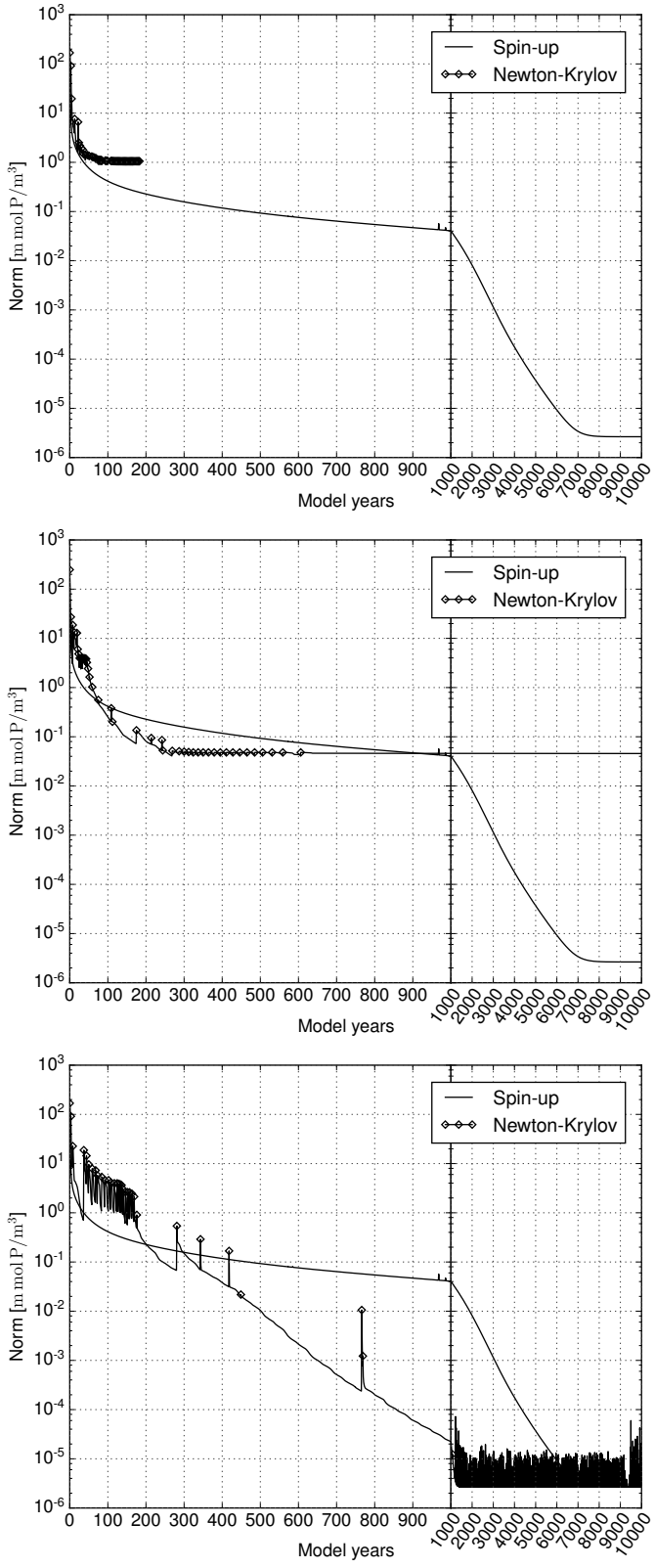

Figure 212. *NPZ-DOP model:* Convergence towards an annual cycle using a spin-up and a Newton-Krylov solver. *Top:* Default Newton-Krylov setting. *Middle:* Initial value altered to $0.5425\ \mathrm{mmol\,P\,m^{-3}}$ for all tracers. *Bottom:* Inner accuracy altered to $\gamma = 0.3$.

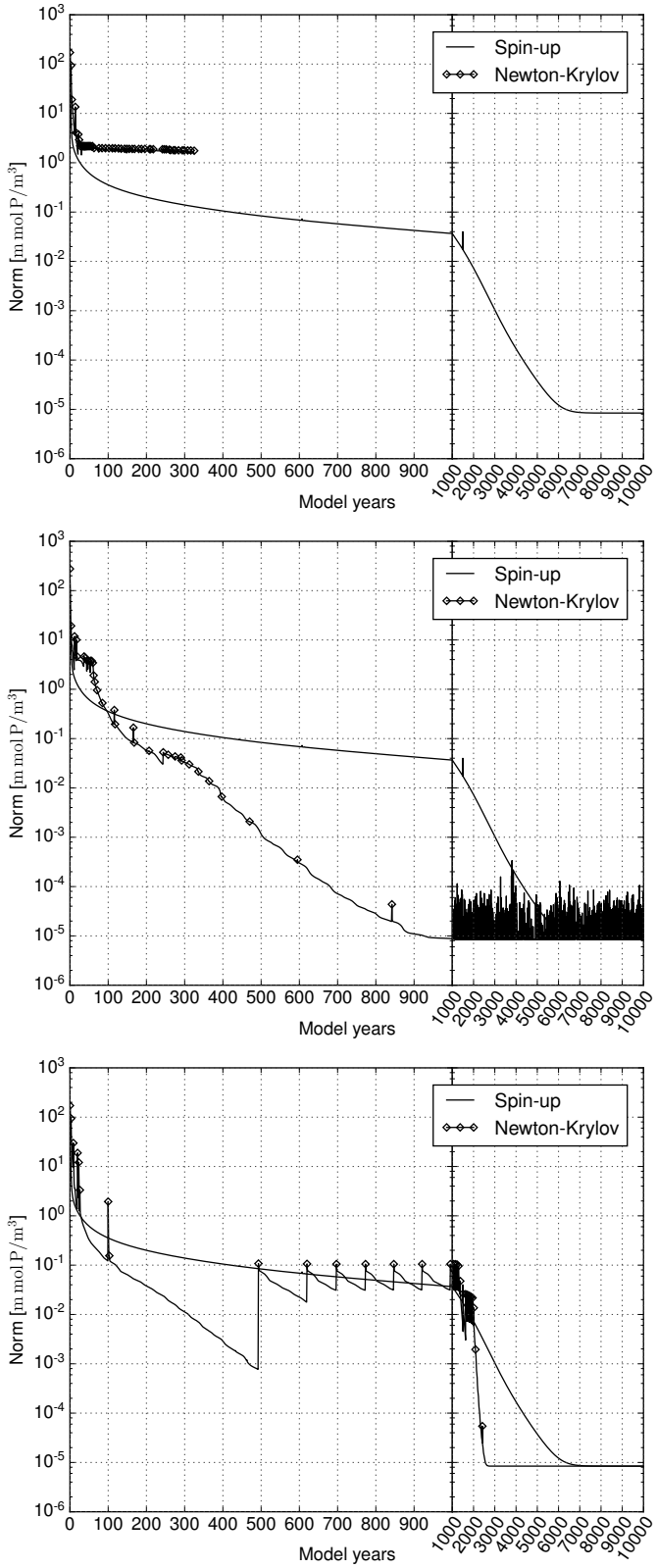

Figure 213. *NPZD-DOP model:* Convergence towards an annual cycle using a spin-up and a Newton-Krylov solver. *Top:* Default Newton-Krylov setting. *Middle:* Initial value altered to $0.0434\ \mathrm{mmol\,P\,m^{-3}}$ for all tracers. *Bottom:* Inner accuracy altered to $\gamma = 0.3$.

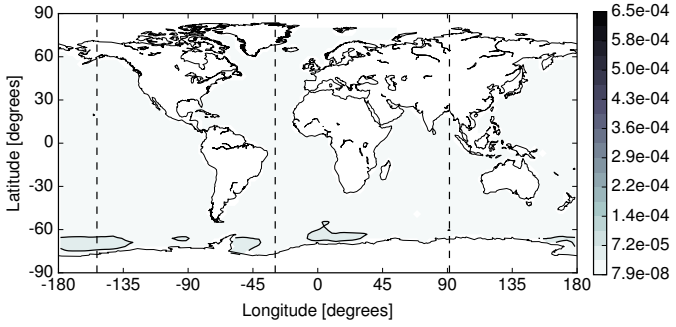

Figure 214. *NPZ-DOP model:* Difference between the phosphate concentration of the spin-up and the Newton solution at the first layer (0 – 50 m) in the Euclidean norm. Units are $\mathrm{mmol\,P\,m^{-3}}$.

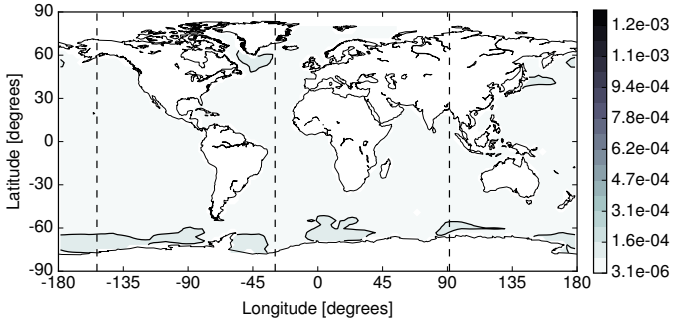

Figure 215. *NPZD-DOP model:* Difference between the phosphate concentration of the spin-up and the Newton solution at the first layer (0 – 50 m) in the Euclidean norm. Units are $\mathrm{mmol\,P\,m^{-3}}$.

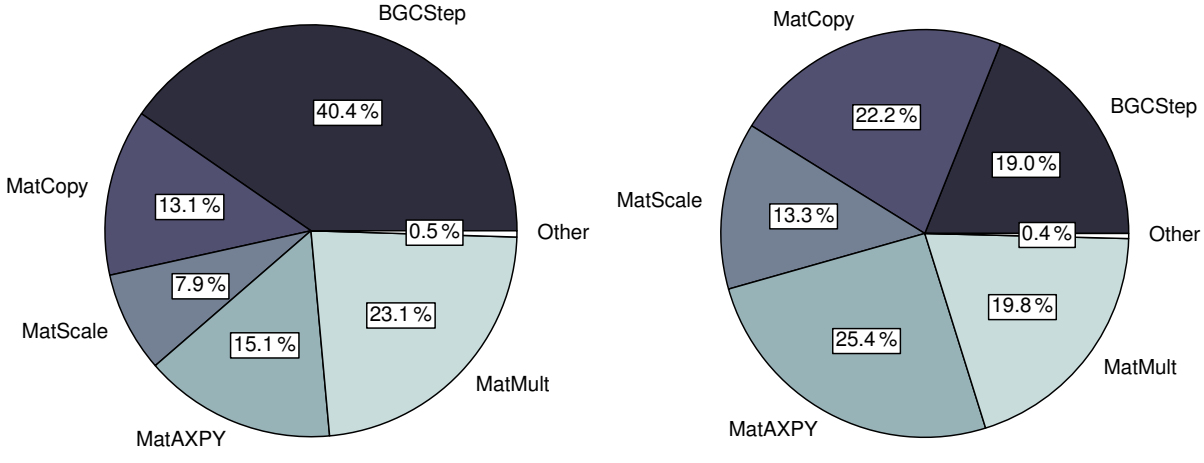

Figure 216. Distribution of computational time among main operations during integration of one model year. *Left:* MITgcm-PO4-DOP model. *Right:* N model.

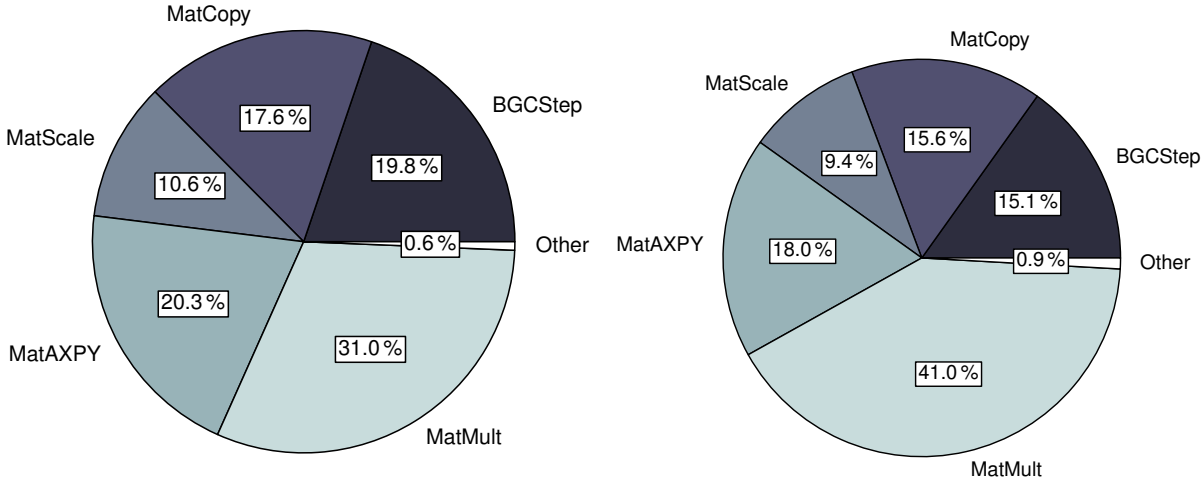

Figure 217. Distribution of computational time among main operations during integration of one model year. *Left:* N-DOP model. *Right:* NP-DOP model.

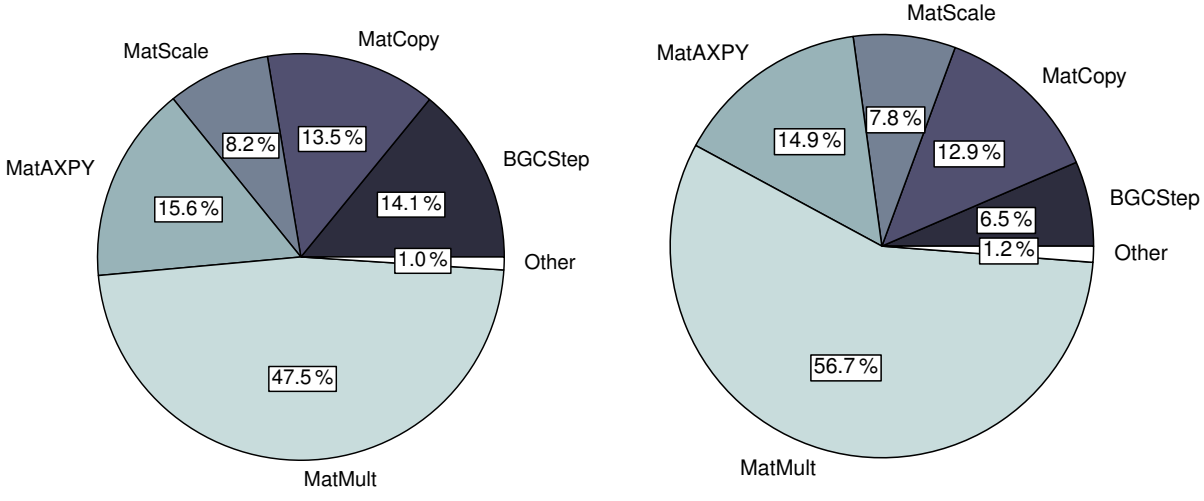

Figure 218. Distribution of computational time among main operations during integration of one model year. *Left:* NPZ-DOP model. *Right:* NPZD-DOP model.

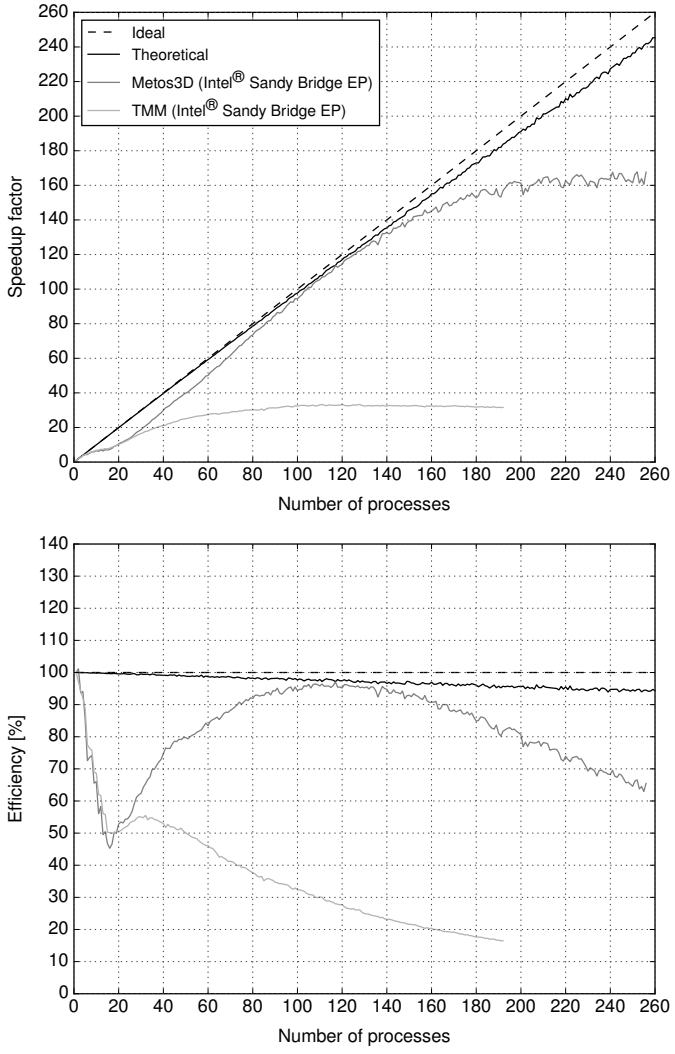

Figure 219. *MITgcm-PO4-DOP model:* Ideal and actual speed-up factors and efficiency of parallelized computations. The term 'theoretical' here refers to the use of load distribution as introduced in Section 6.3.

Algorithm 1: `Phi` (ϕ)

Input : initial condition: $(t_0, \boldsymbol{y}_0)$, time step: Δt, number of time steps: n_t, implicit matrices: $\mathbf{A}_{imp}$, explicit matrices: $\mathbf{A}_{exp}$, parameters: $\boldsymbol{u} \in \mathbb{R}^m$, boundary data: $\boldsymbol{b}$, domain data: $\boldsymbol{d}$

Output: final state: $\boldsymbol{y}_{out}$

$\boldsymbol{y}_{in} = \boldsymbol{y}_0$;
for $j = 1, \ldots, n_t$ **do**
    $t_j = \mod(t_0 + (j-1)\Delta t, 1.0)$;
    $\boldsymbol{y}_{out} = \texttt{PhiStep}(t_j, \Delta t, \mathbf{A}_{imp}, \mathbf{A}_{exp}, \boldsymbol{y}_{in}, \boldsymbol{u}, \boldsymbol{b}, \boldsymbol{d})$;
    $\boldsymbol{y}_{in} = \boldsymbol{y}_{out}$;
end

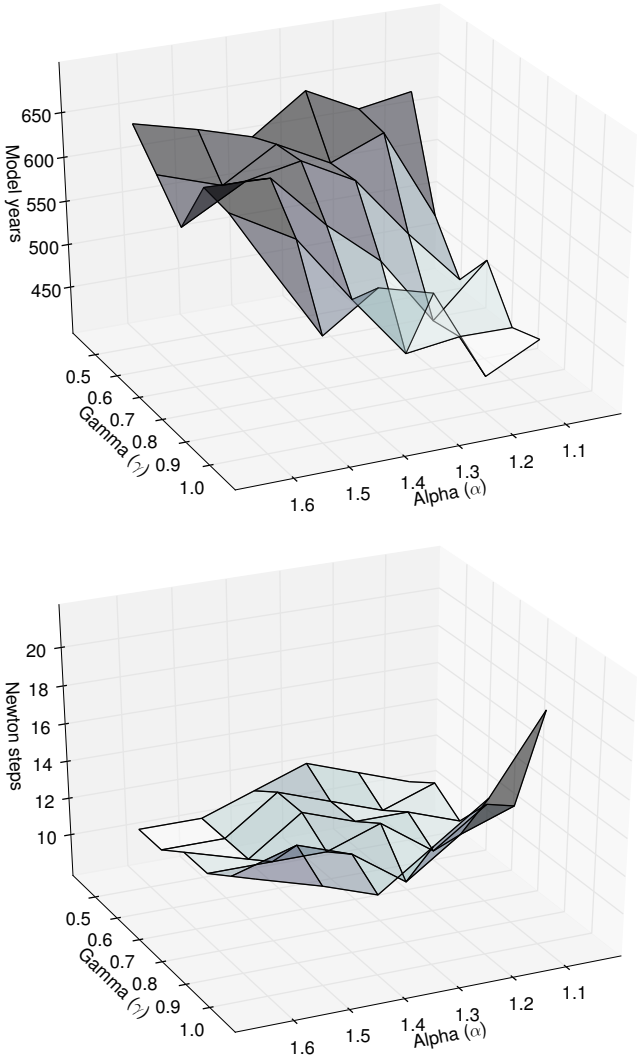

Figure 220. *MITgcm-PO4-DOP model:* Number of model years and Newton steps required for the computation of the annual cycle $\boldsymbol{y}(\boldsymbol{u}_d)$ as a function of different convergence control parameters α and γ (cf. Equation (8)).

Algorithm 2: `PhiStep` (φ_j)

Input : point in time: t_j, time step: Δt, implicit matrices: $\mathbf{A}_{imp}$, explicit matrices: $\mathbf{A}_{exp}$, current state: $\boldsymbol{y}_{in}$, parameters: $\boldsymbol{u} \in \mathbb{R}^m$, boundary data: $\boldsymbol{b}$, domain data: $\boldsymbol{d}$

Output: next state: $\boldsymbol{y}_{out}$

$\boldsymbol{q} = \texttt{BGCStep}(t_j, \Delta t, \boldsymbol{y}_{in}, \boldsymbol{u}, \boldsymbol{b}, \boldsymbol{d})$;

$\boldsymbol{y}_w = \texttt{TransportStep}(t_j, \mathbf{A}_{exp}, \boldsymbol{y}_{in})$;

$\boldsymbol{y}_w = \boldsymbol{y}_w + \boldsymbol{q}$;

$\boldsymbol{y}_{out} = \texttt{TransportStep}(t_j, \mathbf{A}_{imp}, \boldsymbol{y}_w)$;

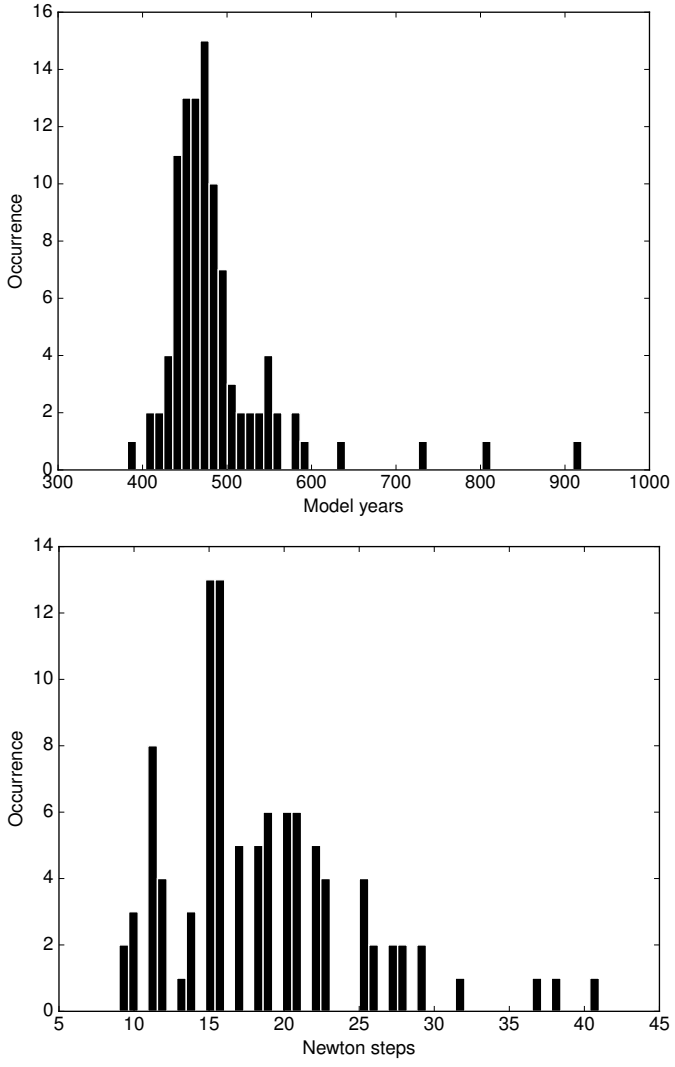

Figure 221. Distribution of number of model years and Newton steps required for the computation of one annual cycle using 100 random parameter samples (cf. Section 6.5).

Algorithm 3: Load balancing

Input : vector length: n_x, number of profiles: n_p, profile lengths: $(n_{x,k})_{k=1}^{n_p}$, number of processes: N
Output: profiles per process: $(n_{p,i})_{i=1}^{N}$

$w = 0$;
$n_{p,1\ldots N} = 0$;
for $k = 1, \ldots, n_p$ **do**
    $i = \text{floor}(((w + 0.5 * n_{x,k})/n_y) * N)$;
    $n_{p,i} = n_{p,i} + 1$;
    $w = w + n_{x,k}$;
end

Table 26. Vertical layers of the numerical model, in meters.

Layer	Depth of layer bottom	Thickness of layer (Δz)
1	50	50
2	120	70
3	220	100
4	360	140
5	550	190
6	790	240
7	1080	290
8	1420	340
9	1810	390
10	2250	440
11	2740	490
12	3280	540
13	3870	590
14	4510	640
15	5200	690

Table 27. Parameters implemented in the MITgcm-PO4-DOP model. Specified are the location within the parameter vector, the symbol used by Dutkiewicz et al. (2005) and the value used for the computation of the reference solution ($\mathbf{u}_d$). Shown are furthermore the lower ($\mathbf{b}_l$) and upper ($\mathbf{b}_u$) boundaries used for the parameter samples experiment.

$\mathbf{u}$	Symbol	$\mathbf{u}_d$	$\mathbf{b}_l$	$\mathbf{b}_u$	Unit
u_1	κ_{remin}	0.5	0.25	0.75	y^{-1}
u_2	α	2.0	1.5	200.0	$\mathrm{mmol\,P\,m}^{-3}$
u_3	f_{DOP}	0.67	0.05	0.95	1
u_4	κ_{PO_4}	0.5	0.25	1.5	$\mathrm{mmol\,P\,m}^{-3}$
u_5	κ_I	30.0	10.0	50.0	$\mathrm{W\,m}^{-1}$
u_6	k	0.02	0.01	0.05	m^{-1}
u_7	a_{remin}	0.858	0.7	1.5	1

Algorithm 4: Interpolation

Input : point in time: $t \in [0,1[$, number of data points: n_{data}
Output: weights: α, β, indices: j_α, j_β

1 $w = t * n_{data} + 0.5$;
2 $\beta = \mathrm{mod}(w, 1.0)$;
3 $j_\beta = \mathrm{mod}(\mathrm{floor}(w), n_{data})$;
4 $\alpha = (1.0 - \beta)$;
5 $j_\alpha = \mathrm{mod}(\mathrm{floor}(w) + n_{data} - 1, n_{data})$;

Listing 1. Fortran 95 implementation of the coupling interface for biogeochemical models.

```
subroutine metos3dbgc(ny, nx, nu, nb, nd, dt, q, t, y, u, b, d)
    integer :: ny, nx, nu, nb, nd
    real*8  :: dt, q(nx, ny), t, y(nx, ny), u(nu), b(nb), d(nx, nd)
end subroutine
```

Table 28. Parameter values used for the solver experiments with the N, N-DOP, NP-DOP, NPZ-DOP and NPZD-DOP model hierarchy.

Parameter	N	N-DOP	NP-DOP	NPZ-DOP	NPZD-DOP	Unit
k_w	0.02	0.02	0.02	0.02	0.02	m^{-1}
k_c			0.48	0.48	0.48	$(\mathrm{mmol\,P\,m}^{-3})^{-1}\mathrm{m}^{-1}$
μ_P	2.0	2.0	2.0	2.0	2.0	d^{-1}
μ_Z			2.0	2.0	2.0	d^{-1}
K_N	0.5	0.5	0.5	0.5	0.5	$\mathrm{mmol\,P\,m}^{-3}$
K_P			0.088	0.088	0.088	$\mathrm{mmol\,P\,m}^{-3}$
K_I	30.0	30.0	30.0	30.0	30.0	$\mathrm{W\,m}^{-2}$
σ_Z				0.75	0.75	1
σ_{DOP}		0.67	0.67	0.67	0.67	1
λ_P			0.04	0.04	0.04	d^{-1}
κ_P			4.0			$(\mathrm{mmol\,P\,m}^{-3})^{-1}\mathrm{d}^{-1}$
λ_Z				0.03	0.03	d^{-1}
κ_Z				3.2	3.2	$(\mathrm{mmol\,P\,m}^{-3})^{-1}\mathrm{d}^{-1}$
λ'_P			0.01	0.01	0.01	d^{-1}
λ'_Z				0.01	0.01	d^{-1}
λ'_D					0.05	d^{-1}
λ'_{DOP}		0.5	0.5	0.5	0.5	y^{-1}
b	0.858	0.858	0.858	0.858		1
a_D					0.058	d^{-1}
b_D					0.0	$\mathrm{m\,d}^{-1}$

Table 29. Difference in the Euclidean ($\|\cdot\|_2$) and volume-weighted ($\|\cdot\|_{2,V}$, cf. Eq. (4)) norms between the spin-up ($\boldsymbol{y}_S$) and the Newton ($\boldsymbol{y}_N$) solution for all models. The total volume of the ocean used here is $V \approx 1.174 \times 10^{18}\,\mathrm{m}^3$. Solutions for models NPZ-DOP and NPZD-DOP were produced by experiments with altered inner accuracy or initial value, respectively.

Model	$\|\boldsymbol{y}_S - \boldsymbol{y}_N\|_2$	$\|\boldsymbol{y}_S - \boldsymbol{y}_N\|_{2,V}$
MITgcm-PO4-DOP	1.460e-01	7.473e+05
N	4.640e-01	2.756e+06
N-DOP	2.421e-01	1.199e+06
NP-DOP	7.013e-02	3.633e+05
NPZ-DOP	1.421e-02	8.514e+04
NPZD-DOP	3.750e-02	2.062e+05

Table 210. Minimum, maximum, average and standard deviation of computational time for one model year as well as the computing time per tracer is shown. All computations were performed on a single core Intel Xeon® E5-2670 CPU at 2.6 GHz.

	Min	Max	Avg	StdDev	Min per tracer
N	112.53 s	112.87 s	112.79 s	0.09	112.53 s
N-DOP	142.96 s	143.30 s	143.12 s	0.11	71.48 s
NP-DOP	160.32 s	161.28 s	160.86 s	0.30	53.44 s
NPZ-DOP	185.46 s	185.70 s	185.53 s	0.07	46.37 s
NPZD-DOP	193.99 s	194.63 s	194.09 s	0.19	38.80 s