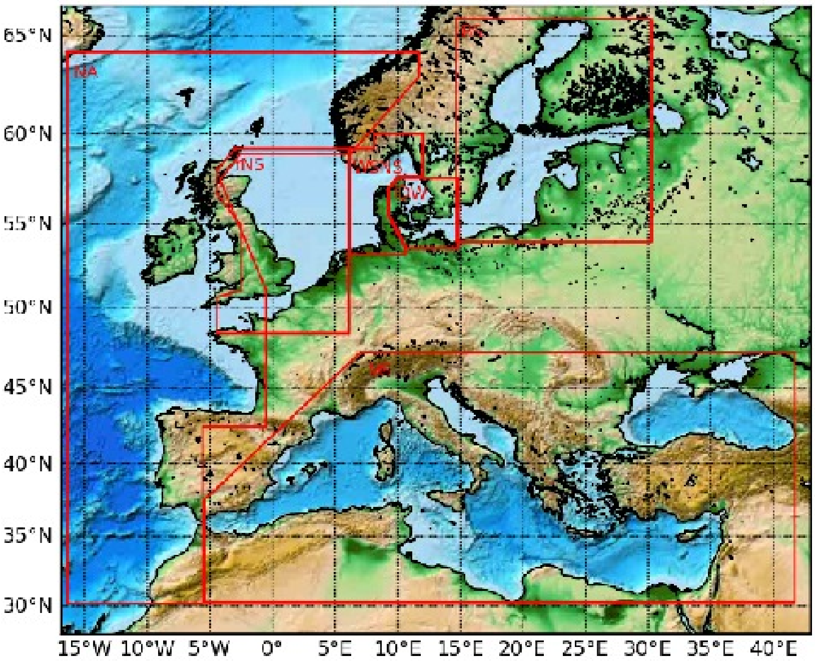
Danish Meteorological Institute
Ministry for Climate and Energy

Technical Report 12-16

More details on HBM - general modelling theory and survey of recent studies

Jacob Weismann Poulsen and Per Berg



www.dmi.dk/dmi/tr12-16.pdf Copenhagen 2012

Colophone

Serial title:
Technical Report 12-16

Title:
More details on HBM - general modelling theory and survey of recent studies

Subtitle:

Authors:
Jacob Weismann Poulsen and Per Berg

Other Contributers:

Responsible Institution:
Danish Meteorological Institute

Language:
English

Keywords:
operational ocean models, PDE, finite difference, stability analysis, CFL, von Neumann, model implementation, numerical stability, floating points, code quality, parallelisation, scaling, reproducibility, validation, verification, IEEE-754, software testing, comparative studies

Url:
www.dmi.dk/dmi/tr12-16.pdf

ISSN:
1399-1388

ISBN:

Version:
1.0

Website:
www.dmi.dk

Copyright:
Danish Meteorological Institute

1 Introduction 3

2 Design goals 5

3 Theory 7
3.1 From blackboard to silicon . . . 8
3.2 Basic solution methods . . . 22
3.3 Musings on time step size selection . . . 23
3.4 Actual time step sizes . . . 26
3.5 Computational intensity . . . 31
3.6 Why is all this important? . . . 33

4 Introduction to the test cases 36
4.1 Benchmark systems . . . 37
4.2 Summary of performance . . . 37
4.3 Summary of performance with passive tracers . . . 39

5 MyO V2.1 41
5.1 Short simulations . . . 43
5.2 Pointwise cross-comparisons . . . 48
5.3 Study of optimization issues . . . 49
5.4 Performance . . . 54
5.5 Longer simulation . . . 56
5.6 The water level in the Baltic Sea . . . 62
5.7 Sea levels around Denmark in 2011 . . . 67
5.8 Storm surge issue in Øresund . . . 71

6 Variant0 74
6.1 Short simulations . . . 75
6.2 Passive tracer performance . . . 75

7 Variant1 82
7.1 Performance profiles . . . 82

8 Variant2 88
8.1 Performance profiles . . . 88

9 Variant3 93

A Appendix: Vector sum using floating points 96

B Appendix: Sea-level maps 102

C Appendix: Abbrevations. 110

DMI
Technical Report 12-16

More details on HBM - general modelling theory and survey of recent studies

Jacob Weismann Poulsen Per Berg

October 9, 2012

1 Introduction

The overall aim behind this paper is to document our *as is* findings with the most recent release of the HBM source code on a number of newly constructed testcases. The 2.6 release of the source code is the first release that has *easy-to-use* MPI support and thus the first release of the source code that is suited for scaling studies. The MPI implementation has by no means been tuned with respect to execution speed and this study will certainly reveal where we should put our efforts on tuning in the future. The sole focus for the current MPI implementation has been on correctness, cf. section 2 on overall design goals for this model.

Moreover, we present a concise survey of the different testcases introducing various numbers aiming at estimating the size of the problem implicitly defined by the testcase at hand and numbers aiming at estimating the expected computational resources required to deal with the testcase. Recently, these definitions gave rise to a broad debate that also touched on how to assess overall model quality. Eventually we decided to expose some of the theoretical foundations for the current implementation allowing a more founded discussion when it comes to cross-comparing different issues, both across different testcases with the same model and across different models. Finally, we decided to spice the theoretical treatment by exposing the different layers that come into play when a model is transferred from a blackboard onto the silicon. We have been urged to present material like this several times and we have used this opportunity to put some of this in writing.

The motivation for gathering the material herein is that the feedback we got and the debate that arose in the slipstream of our recent document [3] indicated to us that more information was needed on both theoretical and practical aspects: These are the two subject we cover here, more theory and more results from model runs, and thus the present paper should be considered a supplement to [3].

We start off by repeating our design goals in section 2. Then, section 3 is devoted to the theory and in different subsections we describe different subjects that one must consider when developing models. This theory section is quite unique in the sense that to our knowledge it is the first time that this kind of material has been gathered and presented as one contiguous document; usually one needs to consult multiple textbooks and papers to retrieve the same information. The theory section can be read alone but it has of course most value when used in conjunction with a model development project. It is our hope that with this theory section we have managed to put sufficient attention on the treated subjects to start a debate on model quality which we believe is needed.

The second part of the present document deals with the results and findings we gathered for a number of real testcases of increasing complexity. In section 4 we give an introduction to the testcases and we summarize and compare the performance characteristics for the testcases to ease decisions and planing related to our future modelling work. There is one section for each testcase, cf. section 5, 6, 7, 8 and 9, respectively, which can be read as self-contained case studies but they are also suited for comparisons. There is naturally much more information available for the first testcase than for the others since we have have had the opportunity to work much more with that one, and together with the investigations shown in [3] on the same test case section 5 demonstrates elements of what we find must necessarily be included in a thorough model verification.

Acknowledgement: We wish to express our gratitude to Maria Grazia Giuffreda from CSCS for granting us access to some of their systems. Finally, we would like to thank Rong Fu, DMI, for helping with some of the plots and tables.

2 Design goals

We will here present (or rather repeat) the overall design goals behind the model implementation before digging into the real stuff. The interested reader is referred to [3] for a thorough description on how we achieve and maintain these properties.

In this implementation of an ocean model

- we rate correctness higher than added functionality,
- we rate optimization with respect to correctness and functionality higher than optimization with respect to speed,
- we rate optimization with respect to speed very high but never so high that we sacrifice portability,

because we strongly believe that this is the right choice in the long run. It is very important that model results are reproducible at any time, anywhere, by anyone.

This implies that

- The code will run in serial (and we must be able to build it on systems that have *no* support of MPI, openMP or openACC).
- The code will run with openMP solely (and we must be able to build it on systems that have *no* support of MPI and openACC). It must be possible to run it with a single thread.
- The code will run with MPI solely (and we must be able to build it on systems that have *no* support of openMP and openACC). It must be possible to run with a single MPI task too.
- The code will run with MPI and openMP. This is the default way of running HBM (and we must be able to build it on systems that have *no* support of openACC). It must be possible to run with a single MPI task and/or a single thread.

The actual decomposition into a number of openMP threads and/or MPI tasks is handled runtime, i.e. requires no re-compiles since each new compilation produces a new executable code which in principle must be tested, so re-compilation means re-validation, and an endless loop has begun.

Moreover, the application must be able to run in all configure incarnations serial, openMP, MPI, and combinations hereof - and produce the exact same results on any given test-case with any number of openMP threads and MPI tasks.

3 Theory

Almost all scientific applications have errors. A sizeable minority (10 %) produces only nonsense.

Brian Vinter, Professor in Computer Science

Here we describe the modelling approach from a more theoretical point of view. The model implementations in the field of atmosphere, climate and ocean modelling tend to be rather involved, rather large and often carry a long development history too. Alas, it is therefore quite common that most modellers do not have time nor interest in the implementation itself and there is an on-going risk that their contributions will be buggy or that their long simulation studies may be inflicted by plain software bugs and hence could potentially be a waste of resources. When we are developing these large models there are things that we can analyse and estimate prior to the implementations and simulations and there are things that are too complicated to be analysed beforehand. Having said that, we feel that it is important that the latter is not used as an excuse for neglecting the former. In general, we strive to be as pragmatic as possible and this basically boils down to something as simple as analysing the components one by one and then eventually combine all of them and study the behaviour by systematic experimentation, i.e. by conducting short and long simulations and explaining the outcome of these. In the present section we will try to convey this idea.

In section 3.1 we attempt to present all the steps that one must go through when going from the ideas on the blackboard until something that ends up in an implementation; the focus in this section will be on analysing the implementation aspects. Then, in section 3.2 we will give a short overview of the basic solution methods that have been chosen for HBM, and in sections 3.3 and 3.4 we discus the computational restrictions these will pose with respect to the time step sizes we can apply. The analysis of the time step sizes is the solution scheme counterpart to the analysis of the underlying algorithms for solving the schemes. It is worth to note that analysis become amenable by doing one component at the time and this approach is used both when it comes to the time step size analysis and the analysis of the algorithms. In section 3.5 we describe how we compare different model setups in terms of how computationally demanding there are. There are pitfalls enough in model development and especially when it comes to model validation and verification the discussion might easily get blurry if we are not careful, so finally, in section 3.6 we dwell some more on why we believe that it all matters, why we should focus that implementation quality.

3.1 From blackboard to silicon

Admitted, there are many rather involved steps in establishing a model and the aim of this section is to try to give a bird's eye view of this whole process. The overall goal being to stress how important each choice along the way is to the final implementation and quality of the model. Moreover, we wish to reveal the many different disciplines that the modelling group have to master in order to reach a decent result in the end. The description of each of these disciplines is overly simplified for obvious reasons but we hope that the readers may find the short appetiser so interesting that some of the references eventually are consulted.

Having said that, we have tried to put slightly more focus on the implementation aspects in this section since in the field of atmosphere and ocean modelling this aspect is rarely described. In this field the essence of modelling is to solve *initial-boundary value problems* for coupled partial differential equations (abbreviated PDE in the following). Thus, in theory one would start off by stating the relevant equations on the blackboard and immediately realises that one can not solve them analytically. Then one rush to the library and finds that there are many ways to approach these problems. First one would have to make a choice on the discretization and consider which of the usual methods (*finite-difference methods*, *finite element methods*, *finite volume methods*, *integral methods*, *boundary element methods*, *spectral methods*, or *pseudo-spectral methods*) that seems most appealing. One will also have to decide on a gridding method where the candidates falls into two categories, namely *structured* or *unstructured* grids. It is important to realize that even at this early stage we are actually making decisions that are likely to impose severe challenges on our final implementation. There are pros and cons for all methods and we are likely forced to make compromises along the way, so we better be aware from an early point of what impacts our choices might imply.

The HBM model chose the finite-difference method, c.f. section 3.2 where we describe the chosen solution in details. At this point of time it is sufficient to note that there are lots of finite-difference schemes to choose from (e.g. *Crank-Nicholson*, *FTCS*, *leapfrog*, *Lax-Wendroff*, ...) and at the early stage of development where things are still being discussed on the blackboard one often aims at neat mathematical properties such as convergence, consistency, stability, good accuracy and so forth when considering the scheme options and it is easy to forget that eventually we will have to implement the scheme and make it work well under very different circumstances than those given in the idealized world of $\mathbb{R}$.

For the sake of completeness we now recall the most basic notions and most important theoretical results (the Lax theorem, the CFL theorem and the von Neumann theorem) pertaining to the field of finite differences. Let $h \in \mathbb{R}_+$ define the size of a *space step* and let $k \in \mathbb{R}_+$ define the size of a *time step*. The pair (h, k) defines a lattice $L_{hk} = h\mathbb{Z} \times k\mathbb{Z} \subseteq \mathbb{R} \times \mathbb{R}$ as the subset of points $(x_m, t_n) = (mh, nk)$ with $m, n \in \mathbb{Z}$. The l_h^2-norm of a grid function $v\colon L_{hk} \to \mathbb{R}$ is the number

$$||v||_h = \left(h \sum_{i=-\infty}^{\infty} |v_i|^2 \right)^{1/2}$$

We will use the symbol l_h^2 to denote the Banach space[1] of grid functions with a finite l_h^2-norm.

For a function $v\colon L_{hk} \to \mathbb{R}$ we let v_m^n denote the value of v at the point (x_m, t_n). A function $v\colon L_{hk} \to \mathbb{R}$ and two sets of constants $\{a_j\}$, $\{b_j\}$ with $a_0 \neq 0$ define a so-called *one-step linear finite difference formula*:

$$\sum_{j=-l}^{q} a_j v_{m+j}^{n+1} = \sum_{j=-l}^{q} b_j v_{m+j}^{n}$$

If $a_j = 0$ for $j \neq 0$ then we say that the formula is *explicit*. Otherwise the formula is *implicit*. Alternatively, we can express the one-step linear finite difference formula as a convolution (with a proper relationship between $\{a_j\}$ and α and $\{b_j\}$ and β):

$$\alpha * v^{n+1} = \beta * v^n$$

One can show that the one-step linear finite difference formula defines a bounded linear operator $S\colon l_h^2 \to l_h^2$ specifying the mapping $v^n \mapsto v^{n+1}$. Using semi-discrete Fourier transforms on the convolutions above we obtain

[1] a complete normed vector space, see e.g. `http://en.wikipedia.org/wiki/Banach_space`

$\hat{\alpha}(\xi)$ and $\hat{\beta}(\xi)$ and we can define the *amplification factor* $g(\xi) = \hat{\beta}(\xi)/\hat{\alpha}(\xi)$. Now, if $v^0 \in l_h^2$ then $v^n = S^n v^0$ and from Fourier analysis results we know that $||v^n|| \leq (||g(\xi)||_\infty)^n \, ||v^0||$. With this background information of finite differences we can present the important results in a compact fashion using standard concepts from functional analysis, see e.g. [14] for a dense presentation of functional analysis.

Let B be a Banach space and let $D\colon B \to B$ be a linear operator. The problem

$$u_t(t) = Du(t),\ 0 \leq t \leq T$$

$$u(0) = u_0$$

where D is fixed but where u_0 may range over all elements in B is called an *initial value problem* and it is said to be *well-posed* if and only if a unique solution $u(t)$ exists for any initial data u_0.

The family of bounded linear operators: $S_k\colon B \to B$ is called the family of *finite differences formulas*. Note that the subscript k implies that the coefficients of the finite difference formula may depend on the time step. We typically see the notation v^n and we use S_k to advance from one step to the next, i.e.

$$v^n = S_k v^{n-1}$$

implying that $v^n = (S_k)^n v^0$. Note that this definition of finite differences formulas encompasses not only one-step explicit formulas but also implicit and multistep formulas. We say that $\{S_k\}$ has *order of accuracy* p if and only if

$$||u(t+k) - S_k u(t)|| = \mathcal{O}(k^{p+1}) \text{ as } k \to 0$$

for any $t \in [0;T]$. We say that $\{S_k\}$ is *consistent* with u if and only if it has order of accuracy $p > 0$. We say that $\{S_k\}$ is *convergent* if and only if

$$\lim_{\substack{k \to 0 \\ nk = t}} ||(S_k)^n u(0) - u(t)|| = 0$$

for any $t \in [0;T]$. We say that $\{S_k\}$ is *stable* if and only if there exits a constant C such that

$$||(S_k)^n|| \leq C$$

for all n and k such that $0 \leq nk \leq T$. The famous theorem due to Lax states that if $\{S_k\}$ is a consistent approximation to a well-posed linear initial value problem then $\{S_k\}$ is convergent if and only if it is stable.

More generally, though, we are dealing with evolution equations formulated as initial-boundary value problems like

$$u_t(x,t) = Du(x,t),\ 0 \leq t \leq T,\ x \in \Omega$$
$$u(x,0) = u_0(x),\ x \in \Omega$$
$$Gu(x,t) = 0,\ x \in \partial\Omega,\ 0 \leq t \leq T$$

where the spatial variable x is confined to a given domain Ω, D is a linear differential operator in x, and G is an operator describing the conditions for u on the boundary $\partial\Omega$ of the domain. For problems in which the initial and boundary data is appropriate to the nature of the differential equation (sometimes known as *properly posed problems*), Lax's theorem states that for a consistent finite difference approximation, stability is the necessary and sufficient condition for convergence (cf. e.g. chapter 14 in [4] for a discussion on finite-difference equations and numerical methods).

That is, the question of convergence boils down to a question of stability and there are two important theorems addressing this issue. These two theorems are also the ones that one turns to when it comes to practical purposes, cf. section 3.2.

The CFL (Courant, Friedrichs and Levy) condition is intuitively reasonable and we will now introduce it and the corresponding theorem in the context of finite difference schemes. Beforehand though, we need to describe dependency domains. The *mathematical domain of dependence* $\mathcal{D}(x,t)$ of $u(x,t)$ is the set of points where the initial data $u(x,0)$ may effect the solution $u(x,t)$. The *numerical domain of dependence* $\mathcal{D}_k(x,t)$ for a fixed value of k is the set of points where the initial data v_j^0 may effect the computation of $v(x,t)$. Let $\mathcal{D}_0(x,t) = \lim_{k\to 0}\mathcal{D}_k(x,t)$. A finite difference scheme is said to satisfy the *CFL condition* if and only if $\mathcal{D}(x,t) \subseteq \mathcal{D}_0(x,t)$ for each pair (x,t). The *CFL theorem* states that the CFL condition is a necessary condition for stability of $\{S_k\}$ with $\{S_k\}$ being a consistent approximation to a well-posed linear initial value problem.

The famous theorem due to von Neumann states that a one-step linear finite difference formula is stable in l_h^2 if and only if the amplification factor satisfy

$$|g(\xi)| \leq 1 + \mathcal{O}(k)$$

as $k \to 0$, uniformly for all $\xi \in [-\pi/h; \pi/h]$. The reader may be familiar with the first part, $|g(\xi)| \leq 1$, which is most often stated in text books and in papers. But, for completeness, when the solution of the PDE is increasing exponentially in time, the necessary and suffcient condition for stability is formally as stated above, i.e. augmenting Mk with M independent of k and h to the right hand side.

It is finally worth mentioning that one never uses the definition of convergence but always stick to either the CFL condition or the von Neumann condition above when doing the analysis, cf. section 3.2. Moreover, it is worth mentioning that instability typically emerges as a local phenomena (but will eventually spread to the global set). This is relevant information when one is to study the outcome of the model. Thus, it can be tricky to distinguish stability issues from parallelizations bugs and nesting bugs which also emerge as local phenomenas, or even from flaws in initial data or boundary data errors. We typically check if the problem moves with different number of tasks and threads (if it does then it is likely a parallelization bug) and we typically study the size of the gradients (if the problem area matches the area with the largest gradients then it is likely a stability issue).

This is classical material and we will refer the reader to the literature for a proper treatment. The introduction above was done solely for the sake of completeness allowing people with different background to get the overall idea without having to consult external references.

As the derivations on the blackboard become more and more concrete, we eventually find that solving quite a few of the complicated PDEs boils down to solve a tridiagonal linear system of equations, cf. section 3.2. In general, boundary value problems usually boil down to solving a large number of simultaneous equations since the boundary conditions must be satisfied simultaneously. In this particular case the set of equations can be described as a tridiagonal system of equations. That is, given vectors a, b, c, d we must find vector s such that

$$\begin{bmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & \cdots & 0 & a_n & b_n \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}$$

or more compactly expressed as

$$a_i s_{i-1} + b_i s_i + c_i s_{i+1} = d_i,\ i = 1, \ldots, n,\ a_1 = 0,\ c_n = 0$$

Generally, when we search for algorithms we focus on various things such as time complexity and space complexity which is often expressed as worst-case complexity using $\mathcal{O}$-notation. It is sometimes convenient to dig deeper and look at the constants (e.g. an $\mathcal{O}(n^2)$ algorithm may be faster than say an $\mathcal{O}(n \log n)$ algorithm for certain types of dataset) and it is sometimes also worth to look at other scenarios than worst-cases, e.g. average-case behaviour or more generally a probabilistic analysis might in some circumstances be more relevant. The complexity in the $\mathcal{O}$-notation is traditionally based on the number of arithmetic operations involved in the algorithm at hand, but very often we face more severe limitations from the memory access pattern of the implemented algorithm than from the actual number of e.g. multiplications and additions involved. Moreover, we need to focus on the inherent dependencies within the algorithm if we are interested in parallelization. In case the algorithm operates on floating point numbers then we must also focus on stability and error bounds, c.f. the following paragraph where we try to introduce some of the relevant concepts for floating point algorithms.

In the case of systems of linear equations we know that we can solve them using Gaussian elimination which have time complexity $\mathcal{O}(n^3)$. Using Gaussian elimination on a tridiagonal system of equations on the other hand boils down to $\mathcal{O}(n)$ operations using double sweeping. The first sweep eliminates the a_i's and a backward substitution then produces the solution, c.f. figure 1. This method will lead to an exact solution in a finite number of arithmetic operations assuming that all operations are done in $\mathbb{R}$ and that the matrix is non-singular. Note that this method is based on the usual LU decomposition where the system $Ms = d$ is rewritten as $LUs = d$ with L being a lower triangular matrix and U being an upper triangular matrix. The system is solved by setting $Us = \alpha$ and then find α such that $L\alpha = d$ and then find s such that $Us = \alpha$. The first sweep decomposes M and solves $Us = \alpha$ and the second sweep solves $L\alpha = d$. This observation is relevant since the error bounds referred below are established for LU decompositions.

It is time to consider how we actually implement an algorithm like the one outlined in figure 1. That is, we will now have to deal with the fact that we move from the idealized world of $\mathbb{R}$ into the rather different world of binary

```
!- forward sweep
e    = b(1)
s(1) = d(1)/e
do i = 2, n
  f(i) = c(i-1)/e
  e    = b(i) - a(i)*f(i)
  s(i) = (d(i) - a(i)*s(i-1))/e
enddo
!- backward sweep
do i = n-1, 1, -1
  s(i) = s(i) - f(i+1)*s(i+1)
enddo
```

Figure 1: An exact algorithm for solving a tridiagonal system as long as all operations are done in $\mathbb{R}$. The method is known as the *double-sweep method*, c.f. section 4.6.3 in [1]. Note that it requires $3n-1$ multiplications and additions and $2n-1$ divisions and it takes $2n$ steps.

floating point numbers. Before we make any comments on the actual implementation we will make a short recap of basic notions within this field. First, we give a short presentation of floating point numbers and then a short presentation of the notions used in the study of floating point algorithms. The references [5], [9], [10] or `http://www.cs.berkeley.edu/~wkahan/` have detailed information on issues related with floating points. Let $\mathbb{F}$ denote a set of floating point numbers with any finite precision and let $\mathbb{F}_n$ be the set of floating point numbers using n-bit precision. In hardware, we will typically find support for $n = 32$ and $n = 64$. The IEEE standard 754 defines how the finite set of bit patterns:

$$\left\{ \underbrace{0\cdots 0}_{n\text{ bits}},\ \underbrace{0\cdots 01}_{n\text{ bits}},\ \underbrace{0\cdots 011}_{n\text{ bits}},\ \ldots,\ \underbrace{1\cdots 1}_{n\text{ bits}} \right\}$$

should be interpreted for $n \in \{32, 64\}$ but the point here is that the set is finite. It is evident from a mathematical point of view that mapping elements $x \mapsto \hat{x}$ from the uncountable set $\mathbb{R}$ into the finite set of floating point numbers $\mathbb{F}$ is bound to fail. That is, no matter how large representation errors $\Delta x = |x - \hat{x}|$ we accept we will still have uncountable many numbers that we cannot represent and most of the numbers that we can indeed represent have a rounding error $\Delta x > 0$ that we must take into account in our analysis. The fact that there are uncountable many numbers that we cannot represent within a given presentation error is not the most critical issue since the range that we can operate in is sufficient for most applications. On the other hand, the fact that all our operations will suffer from rounding errors may give rise to severe obstacles. As a relevant example assume that we would have to do simple statistics such as computing the sum or the mean of the elements of a vector of size N. In appendix A[2] we demonstrate how naive implementations of the sum may provide us with wrong answers and these examples should serve as a warning. If the outcome of the computations matters (which we of course will have all kinds of reasons to believe that it does) then one should care about the way the implementation is done.

Before we head dive into analysing the algorithm itself we will introduce the basic ideas used in this field of constructing stable floating point algorithms. Let $f : \mathbb{R} \mapsto \mathbb{R}$ be a twice continuous differential function and let $y = f(x)$. Let $\hat{y} \in \mathbb{F}$ be the floating point representation of $y \in \mathbb{R}$. The difference $\Delta y = |y - \hat{y}|$ is called the *forward error*. A reasonable question to pose is what other values near x would give rise to the same $\hat{y}$ or rather what is the largest Δx for which $f(x + \Delta x)$ still equals $\hat{y}$. This number Δx is called the *backward error*. When we investigate floating point algorithms we are interested in bounding the backward error and the motivation behind this is that we consider the *rounding errors* as perturbations in the input data. Using Taylor expansion we can express the relationship between changes in input and output as:

$$\frac{\Delta y}{y} = \left(\frac{x f'(x)}{f(x)} \right) \frac{\Delta x}{x} + O(\Delta x)^2$$

The *relative condition number* $c(x) = x f'(x)/f(x)$ of f around x measures the relative change in the output for a given relative change in the input or the relationship between forward and backward errors. If f has a large condition number then a small relative change in the input will produce a large relative change in $f(x)$. Thus, the forward and backward errors are related by the condition number. An example seems appropriate so let $f(x) = \arcsin(x)$. Then for x near 1 we have $\arcsin(x) \approx \pi/2$ and the relative condition number $c(x) = x/(\sqrt{1-x^2} \arcsin(x))$ approaches infinity as x approaches 1. The definitions of forward and backward errors above allow us to define what we call a *stable algorithm* which is quite different from the definition of a difference scheme being stable. An algorithm is

[2] In the testcases described in section 8 and 9 we have $N = 6335114$ for the IDW subdomain so a simple sum will suffer from 6335114 rounding errors and this have severe impact on the final result if we do not try deal with it.

said to be *backward stable* if the backward error is small for all inputs x. Of course, "small" is a relative term and its definition will depend on the context. Often, we want the error to be of the same order as, or perhaps only a few orders of magnitude bigger than the *unit round-off* also known as the *machine epsilon*. The unit round-off is $1/2 * 2^{1-p}$ when operating with precision p so the unit round-off is 2^{-32} for 32-bit precision and 2^{-64} for 64-bit precision. An algorithm is *forward stable* if its forward error divided by the condition number of the problem is small. This means that an algorithm is forward stable if it has a forward error of magnitude similar to some backward stable algorithm. An algorithm is said to be *forward-backward stable* or *mixed stable* or *numerically stable* if it solves a nearby problem approximately, i.e. if there exists a Δx such that both Δx is small and $\Delta y = |f(x + \Delta x) - \hat{y}|$ is small. Hence, a backward stable algorithm is always forward-backward stable. We typically state the mixed stability condition as:

$$\hat{y} + \Delta y = f(x + \Delta x), \ \Delta y \leq \varepsilon_1 |y|, \ \Delta x \leq \varepsilon_2 |x|$$

Let us return to the double-sweep algorithm and recap the stability issues as well as the well-established error bounds on this algorithm. Assuming that there are no restrictions on the input, i.e. on ranges for a, b, c and d. Then there is one obvious problem with the algorithm even when we operate in $\mathbb{R}$, namely that the matrix may be singular leading to zero divisions. Thus, assume that the matrix is non-singular. Then when we move from $\mathbb{R}$ to $\mathbb{F}$ we should be careful that $e = b_i - a_i f_i$ does not round to or is flushed[3] to zero or becomes so small that the division will overflow. It is well-known that this will not happen if the matrix is diagonally dominant, i.e. if $|b_i| > |a_i| + |c_i|$. Thus, when we setup the problem we should ensure that the matrix becomes diagonally dominant to prevent the zero pivots. Initially, this condition was tested at runtime but we now have solid experimental experience that allow us to omit this test and only use it for debugging purposes. If one had the time and patience one could probably also derive this experimental result analytically but we have not done it. However, even for diagonally dominant problems, say $b_i = 1$ and $|a_i| + |c_i| << 1$, there is a risk that we encounter an underflow in the product $a_i f_i$ which we might have to deal with. Actually, this is something we often experience when we cold-start the model. As for the error bounds this algorithm is studied in e.g. [6] where they show that

[3] If e is so small that it has been represented by a denormalized number then depending on compiler options (or platform since some platforms does not support denormalized numbers) it may be set to zero.

for a diagonally dominant matrix M we have:

$$(M + \Delta M)\hat{x} = d,\ |\Delta M| \leq f(u)|M|,\ f(u) = \frac{3(4u + 3u^2 + u^3)}{1 - u}$$

with u being the unit round-off error. There are even more tight bounds in the paper in special cases and more loose bounds when M is not diagonally dominant. Note that the error bound above is independent of the size of the matrix n. If this error bound (or the more tight ones from the paper) seems insufficient for the data sets that we expect then we could try to improve it using *error-free* transformations like it was done for summation in appendix A. To the best of our knowledge such an implementation and a related study has not been published before but [7] have conducted such a study for triangular systems. Another study that we have not yet seen published is what $f(u)$ would look like if one was allowed to use the fuzed operations[4] that was included in the most recent IEEE 754-2008 standard. The bounds in [6] are all established without using fuzed operations which is quite reasonable since they were not part of the standard when the paper was published.

We have now reached a stage where we have an algorithm that provides us with an exact solution in $\mathbb{R}$ and we understand the limitations and error bounds when we implement the algorithm in $\mathbb{F}$, i.e. we know the performance of the solution method with respect to stability and error bounds. Thus, it is time to consider what we can do in terms of performance with respect to computational speed. The algorithm is inherently serial with its loop carried dependencies, i.e. it has built-in execution-order constrains, more specifically *flow* and *anti* dependencies in the first sweep and *flow* dependencies in the second sweep. An algorithm with such constrains cannot be a candidate for instruction level parallelization and vectorization of the loops is out of the question; if we need parallelization we will have to do it on an outer level covering chunks of double-sweep solvers, see e.g. our presentation in chapter 4 of [3]. Thus on our default platform (`x86_64`) we expect to see the computations done with the non-vector SSE2 assembler instructions `divsd,subsd,mulsd` on older `x86_64` without `FMACxx` instructions and with the non-vector assembler instructions `vdivsd,vfnmaddsd` on `x86_64` with the newly introduced `FMACxx` instructions. If written in Fortran90 as outlined in the code snipped in figure 1 then we should probably inline the code manually since compilers will not be able to tell whether or not arrays are

[4] e.g. a fuzed multiply-add operation, aka FMA.

stored with stride-one. Some compilers will generate multiple versions and choose the right one at runtime while others will play safe and assume that arrays are not stride-one. No matter what they choose we will pay a penalty and thus should inline it manually whenever performance with respect to speed matters. The fact that our first choice of algorithm turned out to be inherently serial should not prevent us from moving on but we need to find a new algorithm. This is still an active area of research and in this introduction we will confine ourselves with referring the interested readers to some relevant papers such as [18], [12], [17], [13]. Note that if we change the algorithm then we will have to reconsider stability and error bounds and we might need to do the analysis ourselves or find it elsewhere since it is quite common for the papers that are introducing these new algorithm to deal only with the algorithms in the $\mathbb{R}$ context, c.f. e.g. the reference pair [13], [17].

In the paragraphs above we illustrated the steps that one must follow in order to implement one of the solvers that we need in order to implement this model. Eventually we will put many of these components together and the idealized analysis may no longer hold. Moreover, there are many aspects that one cannot analyse on a piece of paper. For instance, the latest IEEE standard for floating points (IEEE 754-2008) does not specify mandatory requirements for any elementary function such as e^x, $\sin(x)$, x^n, $\ln(x)$ and so forth. The IEEE 754 (and the hardware that we use) allows us to do additions, subtractions, multiplications, divisions and square-roots with correct rounding so the *only* functions that we can represent are those who can be represented using a finite number of these operations. That is, *polynomials* if using only the faster operations (additions, subtractions, multiplications) and *piecewise rational functions* if using divisions too. Thus, elementary functions are implemented (at least for 64-bit where table based method are impossible) using either polynomial[5] or piecewise rational approximations[6]. Again, in theory this sounds sufficient since according to the famous theorem due to Weierstrass (1885) we know that we can approximate any continuous function as accurate as we want using a polynomial (that is using only additions, subtractions and multiplications). Alas, a few years earlier (1882), Lindemann showed that the exponential of an algebraic number (different from zero) is not algebraic and this fact poses a challenge to get the correct

[5] Approximations using for instance Chebyshev, Legrendre, Jacobi or Laguerre polynomials.

[6] Padé approximation is often mentioned in this context.

rounding when doing approximations. We need some definitions to explain what is now known as the *Table Maker's Dilemma.* A number $y \in \mathbb{F}$ is called a *breakpoint* if the rounding changes at y, i.e. if $x < y < z$ then $\hat{x} < \hat{z}$. We can only represent rational numbers $x \in \mathbb{F}$ and since the rational numbers are algebraic we now know that e^x is not algebraic and thus cannot be a breakpoint. Now, assume that $x \in \mathbb{F}$ and that $f : \mathbb{R} \to \mathbb{R}$ is such that $f(x)$ is very close to a breakpoint. The Table Maker's Dilemma is that one cannot say how many bits precision say m one have to have in $f(x)$ to get the rounding correct. Well, since we are operating on a finite set of elements $\mathbb{F}$ we can just for each function f set $m_f = \max m_i(f)$ with $m_i(f)$ being the number of bits required to handle $f(x_i)$ and with i being the enumerator for the elements in $\mathbb{F}$. However, the problem is that we do not know the magnitude of m. It should be mentioned that finding these bounds for important functions is still an active area of research and the latest revision of the IEEE-754 standard (2008) has consequently added recommendations (but not made it mandatory) for rounding of several elementary functions.

The analysis we have done above coincides with the stability analysis that we will do in the next section. By conducting the analysis of each component one by one we allow ourselves to do it with pen and paper but the simplification introduced by chopping up the task into self-contained tasks must be confirmed in the real context and we need to do longer simulations to measure the real effects.

At this point of time the reader may wonder - why do we bother? There are so many factors that may influence the final outcome of our model and it should thus be safe to neglect the issue expressed in the steps above, right? For a start, our initial conditions are not perfect nor are our boundary conditions so even the problem we are trying to solve is somewhat inaccurate. At the end of the day we will compare the solution to this inaccurate presentation of the problem with real observations which by the way are inaccurate too and make a judgement on the result of the model. If we are disappointed about the result then we have to consider how we can improve it and we can try to fiddle with the parameters where parameters in this context have a very broad definition. That is, we can study and try to improve initial conditions, boundary conditions, the discretization (the resolution and the description itself, i.e. the bathymetry), the physical parametrisation and the choice of the values of the corresponding free parameters, ... However, if there are underlying issues in the implementation itself then these studies can be blurred and frankly speaking a waste of time. Our ultimate goal is that these kinds of resource consuming activities (both in terms of man hours and computational resources) are based on a solid foundation so the findings are not obscured by implementation issues. We want the model to reflect the underlying laws of physics through the equations we brought forward on the blackboard, not being dictated by spurious implementation effects. Think about it, why should e.g. the solution to Navier-Stokes equations depend on the number of cores that the modeller had chosen to run the model on?

That is, we encourage that one does try to understand the theoretical results concerning the important algorithms used and that one does systematic software testing which by the way is quite different from cross-comparing observations with model results. A systematic software testing will hopefully reveal plain programming bugs, numerical issues and parallelization issues but just as important it will reveal the magnitude of the uncertainties introduced by compilers, libraries and other system aspects such as the underlying hardware. Once we know this magnitude we also know when it becomes meaningless to adjust the buttons to drag model outcome closer to observations.

One of the arguments that we sometimes meet is that although it is not proved that the atmosphere or the ocean is chaotic there are strong evidence that it is, so we cannot expect identical results when we do simulations. Well, it is important to distinguish between the definitions here. What Lorentz showed was that small differences in the initial conditions will amplify until they are not longer small. If we assume that the initial conditions are not changed then we should still expect that results are reproducible, so running the same model setup twice must lead to identical results, running the same model setup with different decompositions must lead to identical results unless we deliberately have chosen to construct the implementation such that the outcome of the model depends on the chosen decomposition in which case we must in some way or the other verify that this behavior is as intended. The differences in the model results that we may see when running on different platforms are *not* due to the chaotic behaviour (unless we can show that there are differences in the reading in of the initial data or the forcing data) but due to system aspects and possible a buggy implementation in the first place. Hopefully, one can show that the differences found when cross-comparing are in the ballpark of expected system differences but this is our responsibility to investigate this.

Recently, there have also been an increased interest in developing ensemble forecast systems to substitute the single deterministic model integration and the underlying ideas behind this seem very reasonable. Ensemble forecasting is a form of Monte Carlo analysis where multiple numerical predictions are conducted using slightly different initial conditions that are all plausible given the past and current set of observations. Again, this approach sounds very appealing but in our point of view it requires a sound foundation, i.e. that one have conducted proper software testing of the model(s) used. If this is indeed the case then the EPS[7] runs can help forecasters in judging whether the ocean/atmosphere is in a predictable or unpredictable state and they can use it to judge the overall reliability of the forecast. However, if one or more of the EPS runs are based on a fragile foundation then it becomes even harder for the forecasters and others to use the outcome of the system. In our view EPS runs might blur systematic implementation bugs/issues and first become useful when significant efforts have been spend on testing the foundations. This is not to say that we expect that one will have a model without bugs but just to say that we should have put effort into fixing all the obvious bugs and to estimate a confidence level of the model results *before* we try to embrace additional layers of complexity. When it comes down to it, ensemble averaging including just one bug is still a buggy result. EPS is not stronger than the weakest link (unless that was discarded as an outlier).

The same kind of caution should be taken with data assimilation. Both the model and the assimilation procedure must be verified independently and their implementations must be tested for correctness. We have more general comments on data assimilation in the last section of this chapter.

In conclusion, the fact that there are many factors that may influence the final outcome of our model should *not* be used as an excuse for ignoring plain programming bugs, numerical issues and parallelization issues, nor should it become a general disclaimer against the severe, but fortunately rare mistake that the mathematical concept one started off with on the blackboard does not describe the physical problem one wanted to model.

[7] EPS: Ensemble Prediction System.

3.2 Basic solution methods

We will give a brief description of the methods and algorithms we have applied for solving governing equations which are often sets of partial differential equations in space and time, because this might not be evident to the reader and it might be different from what other models do. We will confine ourselves to cover the most computationally expensive parts of the code.

Detailed information on the grid, nesting procedure and data structures is given in [3] and will not be repeated here.

We solve the coupled set of equations of motion and mass equation by a split-step scheme. In the first step, we use the equations of motion to solve for horizontal velocity (u, v) at time step $n + 1$ from knowledge of surface elevation η and (u, v) at the previous time step n and forcing parameters. In the second step, we use the mass equation and (u, v) at the new time step to update η from time step n to $n + 1$. A motivation for this choice and not solving for η, u and v at the same time is that the mass equation is essentially 2D (using depth-integration of the 3D (u, v) field) with only little computational demand, while the equations of motion are in 3D and require some computationally heavy solution methods.

For the equations of motion we apply a vector-upwind scheme for the advective-momentum and cross-momentum terms. All horizontal terms are treated explicitly with a forward-in-time, central-in-space scheme except for the bottom friction term and for the sea ice drag term which have weight 0.5 on both the new and the old time step. Vertical diffusivity of momentum is treated fully implicitly. For each water column, this gives rise to two tri-diagonal linear systems of equations, one for u and one for v, which are solved by the well-known double-sweep algorithm, cf. section 3.1.

Turbulence closure is obtained by use of the Smagorinsky sub-grid-scale model for the horizontal directions and by a two-equation turbulence model in the vertical direction. In the first, the eddy viscosity is obtained by calculating shear and stretch of the resolved horizontal flow (u, v). The latter consists of a k-ω model with algebraic structure functions, cf. [2], and requires solution of two vertical transport equations with source and sink terms; sinks and vertical diffusion are treated implicitly while other terms are explicit. With linearization of non-linear sink terms this results in two tri-diagonal linear systems of equations for each water column which can be solved by standard methods.

Tracer advection is performed with a truly 3D control volume, TVD scheme[8]. Explicit time stepping is applied. To speed up the computations for more than one tracer, we have arranged the data structures and the advection scheme such that the first index of the tracer component array as well as the innermost loops run over the number of tracer components.

Horizontal tracer diffusion is solved explicitly while vertical tracer diffusion is, like for momentum, solved implicitly via a standard tri-diagonal solver. Also for tracer diffusion we have the number of components as the innermost loop.

In all cases, low order finite differencing is applied for spatial derivatives. This is especially important for the horizontal direction where we thereby can maintain short operator lengths of plus/minus one single grid point. As discussed in [3], this eases both serial and parallel optimizations, e.g. MPI communication can take advantage of only one point wide halo zones. The operator length also dictates where proper barriers should be placed in the code, e.g. when halo-swaps between different MPI tasks are required or when explicit barriers inside OpenMP parallel sections are needed to join and fork the threads.

It should by now be clear that solution of tri-diagonal linear system of equations plays a major role in HBM and thus it is justified why we should spend much resources on this subject, cf. section 3.1, and also why we should put effort into tri-diagonal solvers on alternative hardware (cf. e.g. the GPU port we discussed in [3]) and have an eye on the on-going the research in that area.

3.3 Musings on time step size selection

As shown in [3] and also from the cases presented later in the present report (cf. sections 4 - 9) there is a lot we can do with the code as is in terms of domain decomposition and parallelization. Large models in terms of number of grid points do not necessarily take longer time to execute than a small model; it is merely a matter of how many resources in terms of hardware (cores) we choose to spend (within some limits, of course). And we can

[8] TVD scheme: Total Variation Diminishing.

even choose to dedicate development time on optimizing the code further within each time step, e.g. IO parts and MPI communication as well as some of the number-crunching parts certainly has potential for being improved. But, unfortunately, we cannot parallelize the time axis. Time is inherently sequential; we must take time step n before we can start on time step $n + 1$. So, a simulation must be run from start to end, time step by time step in sequence, and we must be careful when selecting time step sizes.

Since we are dealing with explicit time stepping schemes, we are unavoidably facing restrictions on the time step sizes we can apply for stability reasons. Even in the cases where we apply implicit schemes, and thereby by-passing restrictions on stability grounds, we should still keep the time step size within reasonable limits in order to maintain satisfactory accuracy; recall that stability does not guarantee accuracy.

The dilemma is that a smaller time step size leads to more stable and more accurate solutions but require longer wall clock time to complete the simulation. Of course we would like to run our models as fast as possible using as little resources as possible, but we would definitely not risk e.g. a storm surge forecast to blow up during a storm just because we compromised on the time step size. When we are dealing with production models we must know the model specifications (including the time step size) and the simulation time window *a priori*. A typical researcher's strategy of reducing the time step size and make a re-run in case of failure is not an option we can afford when we are running operational models in a tight time window; we can usually only upgrade our operational models at the most once or twice per year. Therefore we must make justified choices of the time step sizes during the development and calibration phases of the modelling project. "Better safe than sorry" is the appropriate approach here.

The actual stability criterion depends on the specific discrete scheme and the specific PDE at hand, but in general terms, the finer the grid resolution the more demanding will the restriction on the time step size be. That is (still in very general terms), for hyperbolic problems (e.g. wave equations and advection) we will set an upper limit on a dimensionless velocity (or CFL number) and have criteria in the form of

$$\frac{V\Delta t}{\Delta l} < f_v$$

while for parabolic problems (e.g. diffusion) we will limit a dimensionless diffusion coefficient and the criteria read something like

$$\frac{D\Delta t}{\Delta l^2} < f_d$$

In the above, Δt is the time step size, V is a velocity scale, and D is a diffusion coefficient. In the denominator Δl is a measure of the size of the applied grid spacing, e.g., for 1D vertical problems $\Delta l = \Delta z$, and for 2D horizontal problems $\Delta l = \sqrt{2/\Delta x^2 + 2/\Delta y^2}$. The limits f_v and f_d are some finite numbers closely related to the specific solver, typically $\mathcal{O}(1)$ for stability of explicit schemes and $\mathcal{O}(10)$ for accuracy of implicit schemes.

For finite difference schemes, it is common practise to try to estimate the stability criteria from e.g. linear stability analysis in the von Neumann sense (cf. e.g. Chapter 19 of [15]) while, even for stable schemes, we must often resort to numerical experimentation to find suitable limits within which we still maintain satisfactory accuracy. It also happens that the stability analysis is simply too complicated to treat analytically due to multi dimensions, multi variables, many terms, non-linear terms, non-constant coefficients, complicated geometry, etc, so the analysis is very much about ”playing safe” and identify worst-case scenarios. Remember, stability is a local feature, and sometimes it is possible to do local linearization and to replace non-constant coefficients by constants locally, etc., and in this way escape the stability analysis which otherwise seems hopeless. In any case, the chosen criteria must prove their worth through numerical experiments with real, full-scale, long-term simulations which cover an appropriate range of situations. For classification of PDEs and introduction to basic finite differencing, cf. a classical text book like [4]. For a thorough introduction to computational fluid dynamics, cf. another classic [1].

Please note that in the type of problems we deal with here, as a rule of thumb, the horizontal velocity scales are most often some orders of magnitudes larger than the vertical velocity scales, and the horizontal grid spacing is in the order of 1000 meters while the vertical grid spacing is 1 meter. Therefore, in practical application, we often find ourselves restricted by the first type of criteria above in the horizontal direction and by the second in the vertical direction. The last one can hit us quite severely due to the squared Δl in the denominator. Our experience is that many modellers sometimes forget this fact, e.g. when increasing the vertical resolution from say 4 m to 1 m with a required decrease in time step size by a factor of 16. This should also be kept in mind when comparing different model setups, for example through their computational intensity, cf. section 3.5.

3.4 Actual time step sizes

We do not make an attempt at performing stability analysis of the full set of discretized Navier-Stokes equations; there are simply too many variables, terms and varying coefficients. Instead we resort to term-by-term worst-case analysis which results in simple, practical guidelines for the selection of time step sizes. Even if we could perform analyses on the complete set of finite difference equations in idealized configurations with say flat bottom and constant coefficients, that would likely not tell us more.

Horizontal Smagorinsky terms:
The concept of the Smagorinsky sub-grid-scale model is to act as resistance against shear and stretch deformations of the flow. An eddy viscosity, E_h, is calculated from stretch and shear of the resolved horizontal flow

$$E_h = L^2 \sqrt{\left(\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)^2}$$

where L is the length scale given by

$$L^2 = C_{smag}^2 \Delta x \Delta y \text{ with } C_{smag} = 0.2$$

Analysing separately for the Smagorinsky terms we are facing a parabolic problem which, when discretized into a so-called FTCS[9] scheme, should satisfy

$$E_h \Delta t \left(\frac{2}{\Delta x_i^2} + \frac{2}{\Delta y^2}\right) < 1$$

according to linear stability analysis in von Neumann sense. We should never run into troubles with the Smagorinsky model, not even if more diffusive terms are present, so we implemented a runtime stability assurance that keeps E_h sufficiently low, i.e.

$$E_h < \frac{0.1}{\Delta t \left(\frac{1}{\Delta x_i^2} + \frac{1}{\Delta y^2}\right)}$$

everywhere. Our model calibration must confirm if there is a need for larger E_h, possibly through lower Δt.

[9] FTCS: Forward-in-Time, Central-in-Space.

Background diffusion:

The model has an artificial Laplacian diffusion augmented to the equations of motion. Linear stability analysis for this term alone shows that our scheme is *conditionally stable* provided that

$$D_L \Delta t \left(\frac{2}{\Delta x_i^2} + \frac{2}{\Delta y^2} \right) < 1$$

The intention is to have a tiny background diffusion acting as a stabilizer so a criterion of this kind must under no circumstances become a problem because then the whole idea of having it is destroyed. However, in certain situations we have seen that a fixed value of D_L could lead to spurious problems if the modeller is not cautious. For the so-called *artificial viscosity method* cf. a text book like chapter 14 in [4].

Barotropic, free surface mode:

Here we reduce the stability analysis to the well-known 2D, depth-averaged wave equation at constant depth H:

$$\frac{\partial \eta}{\partial t} = -H \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right)$$

$$\frac{\partial u}{\partial t} = -g \frac{\partial \eta}{\partial x} \, , \; \frac{\partial v}{\partial t} = -g \frac{\partial \eta}{\partial y}$$

When discretized in a straight-forward way and solved with our split-step scheme these are turned into a FTCS scheme which is known to be *unconditionally unstable*, and we therefore need some kind of stabilizer. Fortunately, in the full system of equations we do have stabilizing terms like bottom friction and conditionally stable background diffusion and Smagorinsky terms, see above. It can be shown, that if we augment any kind of diffusive term with coefficient D to the right hand side of the equations of motion, and apply a scheme that in itself is stable, i.e. obey

$$D \Delta t \left(\frac{2}{\Delta x_i^2} + \frac{2}{\Delta y^2} \right) < 1$$

which is the case with both the Smagorinsky model and the background Laplacian diffusion, then our split-step FTCS scheme is *conditionally stable* provided that the following CFL criterion is fulfilled:

$$\mathrm{CFL} = C_{i,j} \Delta t \sqrt{\frac{2}{\Delta x_i^2} + \frac{2}{\Delta y^2}} < 1$$

where the celerity of gravity waves is given by

$$C_{i,j} = \sqrt{gH_{i,j}}$$

It is the location (i, j) where we find the largest value of CFL that determines stability and accuracy. During model initializations a check is performed to verify that this CFL criterion is fulfilled everywhere in every domain.

Flow at high Froude number:
It is quite common in the oceanographic community to focus on Δt for the barotropic, free surface mode. This is quite reasonable and natural since ocean circulation models most often operates on open waters and deals with flows at relatively low Froude numbers, i.e. sub-critical flow

$$F_r = \frac{|\vec{u}|}{C} < 1$$

where the free surface mode is the fastest moving mode. But for flows with large gradients, e.g. with flooding-drying or with steep bottom topography, we can have near-critical, $F_r \sim 1$, or even super-critical flow, $F_r > 1$, and in such cases the momentum advective terms become dominating. We cannot exclude these situations from our operational applications since they are likely occurring e.g. with flooding/drying in the Wadden Sea and for low-water flows over sills. Fortunately, we are saved by the applied upwinding which has been shown to be crucially important in such situations, cf. [8]. If we analyse the applied forward-in-time, upwind-in-space scheme we arrive at

$$\Delta t \left(\frac{|u|}{\Delta x} + \frac{|v|}{\Delta y} \right) < 1$$

which we also need to take into account. There is no built-in check for this criterion in the code so it is entirely up to the user during his/her model calibration to ensure that it is fulfilled.

Vertical diffusion:
As described in the previous section 3.2 we apply an implicit scheme whenever we encounter vertical diffusion terms. Even though the scheme is *unconditionally stable* and is renowned for its relatively inexpensive high-order accuracy (cf. e.g. sections 4.6-9 in [1]) numerical experimentation shows that the solution can be very much off with huge, nasty wiggles if the dimensionless diffusion coefficient is not kept sufficiently small, say less than something between 10 and 15. We here apply the rather conservative limit

$$\frac{D\Delta t}{\Delta z^2} \leq 10$$

It is the location (i, j, k) with the largest value of $D_{i,j,k}/\Delta z^2_{i,j,k}$ that determines Δt. However, we do not know the maximum value of $D_{i,j,k}$ before we run the model because this is obtained from model prognostics calculated through the mixing scheme, cf. [2], but we do know that $D_{i,j,k}$ locally can become quite large for a short period, e.g. due to strong surface cooling. Therefore we stick to a practical approach and put a limit on $D_{i,j,k}$ such that the above criterion is always satisfied; our model calibration must then show us if we resolve processes sufficiently well or if we need smaller Δt to open up for larger D.

3D tracer advection:
Choosing the time step size for tracer advection follows the principle that "you cannot cut a bald head", which here means that during a time step we cannot empty the control volume (i.e. the grid cell) for more substance than it contains; it is physical and numerical nonsense in this context to operate with negative tracer concentrations or negative cell volumes or things like that. Thus, for stability we must assure that the transport *out* of each grid cell during one time step will never exceed what that grid cell contained at the beginning of the time step; this can also be formulated in a CFL-like way. That is, say, for the sake of simplicity, that the local flow is such that there is a transport into the considered grid cell across all grid cell faces except the eastward cell face where the transport is out. The grid cell volume is $h\Delta x\Delta y$ and the outward transport is $h_x u\Delta y\Delta t$ with $u > 0$ in this example. Then, the stability criterion for this situation becomes:

$$\frac{(h_x u)_{i,j,k}\Delta t}{h_{i,j,k}\Delta x_i} < 1$$

This formulation can be more restrictive than the classical CFL criterion for equidistant grids of constant cell height, i.e. $u\Delta t/\Delta x < 1$, which states that the signal is only allowed to propagate less than one grid spacing per time step. In our case the cell face height, h_x, can locally be significantly larger than the centre cell height, h, during a period with pronounced low water (e.g. a storm) and/or near steep bottom topography. We do not know the sea surface variations and the current speeds a priori so we must here rely on our knowledge of the model physics, i.e. we must gather enough information through e.g. long simulations to evaluate a proper value for the maximum allowable Δt for each testcase. We must of course generally take into account transport across all six grid cell faces, not only a single one, by augmenting the left hand side of the above by $(h_y v)_{i-1,j,k}\Delta t/h_{i,j,k}/\Delta y$ if, for example, we also have transport out of the northern face of the grid cell. We have implemented a method that checks for violation of the stability criterion for tracer advection locally at each grid cell during runtime; this is a valuable tool during the development and calibration phases to aid finding a proper Δt and do fine-tuning of the grid (through $h_{i,j,k}$); but during operational runs it should not come into play.

Horizontal tracer diffusion:
Horizontal tracer diffusion is calculated from an effective horizontal eddy diffusivity, D_{eff}, which is obtained from the horizontal Smagorinsky eddy viscosity for momentum, E_h, by a relaxation according to:

$$D_{eff} = \frac{0.25E_h}{0.1 + E_h \Delta t / L^2}$$

The stability criterion for the 2D, explicit diffusion scheme applied here, assuming locally constant D_{eff}, is:

$$D_{eff,i,j} \Delta t \left(\frac{2}{\Delta x_i^2} + \frac{2}{\Delta y^2} \right) < 1$$

so with the above expressions we get a maximum value

$$D_{eff,max} = \frac{0.01 \Delta x_i \Delta y}{\Delta t}$$

which then yields an *unconditionally stable* scheme for horizontal tracers diffusion. And just as for vertical diffusion our model calibration must show if processes are resolved sufficiently well or if a smaller Δt is needed to open up for larger D_{eff}.

Special cases and other considerations:
For the 2D cases above, the stability criteria have all been developed from idealized, smooth configurations. Usually, criteria based on 2D and 3D formulations are relatively more restrictive than their 1D counterparts (e.g. the criterion for diffusion is $\Delta t < 0.5 \Delta x^2 / D$ in 1D while it is $\Delta t < 0.25 \Delta x^2 / D$ in 2D with $\Delta y = \Delta x$), so we are on the safe side. This also holds if we consider more odd layout of the grid (e.g. with corners and narrow channels to describe changes in topography), the criteria obtained from analysis of full 2D or 3D equations reflect the worst-case scenario.

We have treated the equations that are most fundamental to our present model implementation, and these are also those that constitute the most time consuming parts of the model and those parts that usually put the most severe restrictions on the time step sizes we can apply. However, we do at this point need to stress that our presentation here does not cover every single component of the full model system, and occasionally the we see model applications that run into trouble due to violated restrictions in other model components, e.g. when including an add-on ecological model (which is passive to the physics) using explicit time stepping under conditions with large sink terms.

Finally, we must make a remark regarding the prescribed initial and boundary conditions. All the way through this theory chapter we have taken the pragmatic attitude towards the available initial and boundary data that these reflect the nature of the physical problem at hand and thus are appropriate to the corresponding PDE and its discretization. There is, unfortunately, no guarantee that this assumption will always hold since these data are often collected from external sources (i.e. observations or a third-party's model product), and troubles due to e.g. inadequate forcing at the open model boundaries might be confused with a stability problem of the finite difference scheme. In such cases the medicine is not to lower the time step size or increase coefficients which have smoothing effects (such as friction and diffusion parameters). Rather, as a pre-processing step, the initial and boundary data should be screened and prepared thoroughly.

3.5 Computational intensity

This paper will describe testcases of increasing complexity in terms of computational resources. We have tried to come up with a simple crude estimate of the complexity of a testcase. A very simplified description of the problem is that we need to solve a couple of PDEs and the discretization of the space and the time thus defines the size of the overall computation problem. However, we are solving PDEs in both 2D and 3D so should we account for both types? We are highly dominated by the 3D problems so a first estimate of the *computational intensity* I for a model setup would be the sum over all nested areas of the number of 3D water points in that area divided by the time step size (in seconds) used in that area, i.e.

$$I = \sum_{ia=1}^{narea} iw3(ia)/dt(ia)$$

The model at hand is a hydrostatic model and we have 7 prognostic variables (u, v, s, t, tke, diss, avv). The definition above will not be useful in answering questions like how expensive would it be to deal with a non-hydrostatic formulation or say to run with 9 or 25 additional passive tracers. Thus, in order to take additional 3D PDEs into account we need to refine the previous definition with (ensuring that $I = I_v(0)$):

$$I_v(n) = (n+7) * \sum_{ia=1}^{narea} iw3(ia)/dt(ia)$$

A non-hydrostatic formulation would have 9 prognostic variables (the previous 7 plus w, p) so the definition above allows us to estimate that.

To ease comparison between the computation requirements of different models and different setups, it is useful to look at the relative computational intensity, I_r and I_{rv}, which is the intensity of the model setup in question normalized by the intensity of a well-known model setup. Here, we choose to normalize by the specific computational intensity of a certain test-case[10] that has been commonly used at DMI for quick tests as well as longer model simulations (climate studies, re-analysis). This test-case has $I = 10552.5$ sec^{-1}.

Note that we do *not* wish to add any diagnostic variables to the intensity numbers since we regards these as temporary computations required by the overall solution method (e.g. w) or as computations that could be done offline afterwards. We have not even tried to take the fact that some areas will update some variables (u, v, z) twice per computational cycle and others (s, t, n*passive, tke, diss, avv) only every second computational cycle into account. That is, our definition of I_r and I_{rv} is nothing but a first crude estimate of the size of the problem. Having said that, it has been a quite useful working estimate for us, though.

Needless to mention there are two parameters that one can adjust to get a higher I_r number but the two parameters are not totally independent. One can either increase the number of wetpoints or decrease the time steps used but it should be stressed that while increasing the number of wetpoint can be hidden by adding more resources into the solution (say more compute nodes) this is not the case with the time steps. The only parallelism that we have across time steps is the asynchronous parallelism and that is quite limited to things like IO.

[10] This test-case is formally known as the test03 test-case.

It should be stressed that the complexity number is not suited for cross-comparing fundamentally different approaches to solving the PDEs. For instance, a purely implicit scheme is not bound by the same stability criteria as a simple explicit scheme and one would thus end up with very different time step sizes and thus very different complexity measures.

3.6 Why is all this important?

Let us summarize what it involves to develop a model:

A We have identified a physical problem (PP) that is diagnosed through a physical quantity Q (or a set of quantities).

B We develop/select a set of equations that describe the problem mathematically, e.g. a set of PDEs that is meant to describe PP as an initial-boundary value problem (IBVP) in $\mathbb{R}$, such that if we know the initial and boundary values then we can describe the behavior of Q through the PDE.

C We develop/select a solution method, e.g. a finite-difference scheme (FDS) to approximate the PDE and we make sure that the FDS has all the necessary properties (convergence, consistency, stability, accuracy), still working in $\mathbb{R}$.

D We implement the FDS in $\mathbb{F}_n$ using a decent programming language and various kinds of hardware and software features (including parallelization, vectorization, FMAs, etc.) and use this implementation to predict an estimate for Q in $\mathbb{F}_n$, called Q_n.

It is not necessarily a good idea to evaluate the model from observations Q_{obs} of the quantity if that is the only thing one does. Traditionally, there is a tendency in the ocean modelling community to decide the quality of the model by performing raw comparisons of Q_{obs} and Q_n. But, unfortunately, there can be quite a distance between a recording of Q_{obs} from the nature to the Q_n one gets from the model.

- First of all, the Q_{obs} is *not* at all a solution to our IBVP, and it is not even certain that Q_{obs} describes the PP we started out with (due to e.g. instrument failure). If Q_{obs} is meant to enter the evaluation process, it must be ensured that it describes a quantity that is actually modelled and that it does not include effects that are not part of the model. The observations to be used in the evaluation process must undergo a thorough quality assurance process, just like we require for the model.

- Secondly, one must ensure that each of the single steps A-D above are correct individually, i.e. that the IBVP really describes the PP, that the FDS really is a useful approximation to to the PDE, that the implementation into the world of $\mathbb{F}_n$ is in order. One cannot skip say the verification of the implementation and just settle with a quick textbook-lookup insurance of point B or C.

Not that this document should turn into sharing of war stories, but we do actually have examples of operational model components failing in all of the above-mentioned steps A-D, making it very little useful for the public as well as the scientific society, cluttering the discussion and the decision making, and eventually wasting a lot of valuable man-time and project money. This is why we maintain our design goals, this is why we put effort into describing what we do when striving towards our goals. We are not saying that a single person should be responsible for carrying out all the analysis and development through to the end, but at the very least it should be documented what has been done in order to verify the quality.

In this context, it seems reasonable to add a final comment on the use of data assimilation in this area. In theory, data assimilation solves a well-defined problem and the outcome of using it is meant to please those mainly interested in verification scores, i.e. how close the model outcome is to the observations measured. In practice, it is not that simple. Let us summarize the underlying assumptions that must hold:

- The set of observations does indeed describe the PP that the model (in terms of the PDEs) is meant to describe.
- Observations are always correct, i.e. a better verification score is *by-definition* an improvement.
- It is possible to setup automatic procedures that will do perfect quality assurance of the observations (and still have a non-empty set of observations left to assimilate).
- Time interpolation or time nudging of observations does not pose a problem to us (assuming that we are not trying to do 4dvar).

- Uni-variate analysis does not pose a problem to us, e.g. we can adjust say the T field as we please without introducing problems with the state of any of the other variables, say S. Basically assuming that there is no relationship between the variables in our set of prognostic variables and thus basically contradicting the underlying assumption behind the model.

- The fact that the result of the data assimilation is not a model-state does not pose a problem to the model.

- There is not always a one-to-one mapping between an observation and a related prognostic variable in the model. In these cases we assume that we can construct an exact and flawless mapping that allow us to use the observation.

There are several examples in this paper showing that some of these assumptions are extremely hard to justify in real life. Finally, it should be stressed that we do *not* try to say that data assimilation is not useful. All we say is that one should be careful not to focus too much on verification scores solely.

4 Introduction to the test cases

Table 1 summarizes the setups that we describe in the following sections. These range from present day's operational setup, MyO v2.1, to setups we aim at running in the future. Note that the largest problem in the table is 80 times harder than the one we run in production today. With this range of applications at hand we feel that we have prepared a solid foundation for decision making on future operational setups.

Testcase	**3D grid points**	**iw3**	f_{iw3}	**dt**	$I_{rv}(0)$	$I_{rv}(9)$	$I_{rv}(25)$
MyO v2.1	23357904	2838057	12.5%	15	14.3	32.7	51.1
variant0	68435880	8279907	12.1%	12.5	60.6	137.1	214.3
variant1	74560380	9667032	13.0%	10	89.6	204.8	320.0
variant2	129061336	14418360	11.2%	5	227.7	540.5	813.2
variant3	165983872	27554107	16.6%	5	321.2	734.2	1147.1

Table 1: The test cases described in the present report. The time step size **dt** is the smallest used in these nested setups. The number $iw3$ is the number of 3D wetpoints whereas f_{iw3} expresses the fraction of all the 3D points that are indeed wetpoints. Please consult the appropriate tables in the respective sections to see the specific time step size of each of the nested domains.

The difference between the MyO v2.1 and the variant0 case is the horizontal resolution in the Baltic Sea domain which is 3 n.m. and 1 n.m., respectively. The 1 n.m. resolution in the variant0 case puts a more demanding restriction on the time step size. Also, there is slightly more layers in the BS domain of variant0.

Compared to variant0, variant1 has the 1 n.m. WS domain extended to cover Skagerrak which, due to the deep Norwegian Trench requires a lower time step size. In this extended WSNS domain the vertical resolution is also increased. The NS domain is of course reduced in variant1, and we call this domain rNS, see e.g. table 23.

In variant2 we have increased the horizontal resolution in rNS to 1 n.m., and in IDW we have increased it to 0.25 n.m. which requires a time step size of 5 seconds, see e.g. table 25.

The final variant3 setup is a large pan-European setup. It includes variant2 to cover the North Sea - Baltic Sea region, but has an enclosing North Atlantic domain which also nests to a Mediterranean Sea domain, see e.g. figure 41 and table 26.

Please note that even though the timings and profiles for the different setups we show in this report have indeed really been attained on the machines as described using the exact same source code (release 2.6), it is very likely that we will learn from future model analysis and evaluations, as described in section 3.4 that we will have to decrease the time step size for e.g. tracer advection and/or diffusion and/or turbulence. Likewise, it is indeed possible that new, smarter features will be implemented which also affect timings and profiles.

It should further be noted, though this should by now be considered trivial, that the code itself as well as runs of each of the shown setups has passed our usual testing procedure, i.e. checking ANSI compliance, for stack and bounds violations, for safe parallel implementations. Thus, the results for each point along each shown scaling graph are binary identical.

4.1 Benchmark systems

Throughout this paper we will present results from runs conducted on two cray systems, namely our local cray XT5 (12-way cc-NUMA node with 16Gb shared memory on most nodes and 8 nodes with 32Gb, each socket is equipped with Istanbul AMD Opteron at 2.4 GHz) and the cray XE6 system at CSCS (32-way cc-NUMA node with 32Gb shared memory, each socket is equipped with AMD Opteron 6272 2.1 GHz Interlagos processors).

4.2 Summary of performance

Table 5 summarizes the performance attained on our local XT5 system. The I_r ratio between the cases is not reflected in the sustained timings but it should also be kept in mind that we use a different number of cores to reach the sustained performance in the table. Note that we have excluded the variant3 setup. This is *not* because we cannot run it on our local system but we need the use the nodes with 32GB of memory. Alas, we only have 8 of these nodes so we cannot do a relevant scaling study on our local system nor get a number for a reasonable sustained performance. Table 6 presents

Compiler	IEEE flags
cray / 7.4.2.106	-O1 -Ofp0 -K trap=fp
intel / 12.0.4.191	-O0 -traceback -fp-model precise -fp-stack-check -fpe0
gfortran / 4.5.3	-fsignaling-nans -ftrapping-math -fno-unsafe-math-optimizations
pathscale / 3.2.99	-fno-unsafe-math-optimizations -OPT:IEEE_arithmetic,IEEE_arith=1 -TENV:simd_imask=OFF
pgi / 11.10.0	-O0 -Kieee -Ktrap=fp -Mchkstk -Mchkfpstk -Mnoflushz -Mnofpapprox -Mnofprelaxed

Table 2: List of compilers on the XT5 system and the corresponding compiler flags used for IEEE builds in this study.

Compiler	TUNE flags
cray / 7.4.2.106	-O2 -eo -Oipa5
intel / 12.0.4.191	-O3 -fno-alias -ipo -static
gfortran / 4.5.3	-O3 -funroll-loops -ffast-math -finline-functions -finline-limit=5000
pathscale / 3.2.99	-O3
pgi / 11.10.0	-fastsse -Mipa=fast,inline

Table 3: List of compilers used on the cray XT5 system at DMI and the corresponding compiler flags used for TUNE builds.

Compiler	TUNE flags
cray / 8.0.1	-O2 -eo
intel / 12.0.3.174	-O3
gfortran / 4.5.3	-O3 -funroll-loops -ffast-math
pathscale / 4.0.12.1	-O3
pgi / 11.10.0	-O3 -Mipa=inline

Table 4: List of compilers used on the cray XE6 system at CSCS and the corresponding compiler flags used for TUNE builds.

the numbers attained on the cray XE6 system at CSCS. Note that they have 32GB of memory on every node so it is indeed possible to run also the larger variant3 case at scale.

XT5, DMI	**variant0**	**variant1**	**variant2**
24 hour forecast	$<$ 11 minutes	$<$ 18 minutes	$<$ 60 minutes
10 days forecast	$<$ 120 minutes	$<$ 180 minutes	$<$ 10 hours
1 year simulation	$\approx$ 3 days	$\approx$ 4.5 days	$\approx$ 15 days
1 decade simulation	$\approx$ 30 days	$\approx$ 45 days	$\approx$ 150 days

Table 5: Sustained performance on our local cray XT5 system. Note that we use a different number of cores to reach the sustained performance in each of the cases.

XE6, CSCS	**variant0**	**variant1**	**variant2**	**variant3**
24 hour forecast	$<$ 8 minutes	$<$ 15 minutes	$<$ 44 minutes	$<$ 60 minutes
10 days forecast	$<$ 74 minutes	$<$ 142 minutes	$<$ 7.3 hours	$<$ 10 hours
1 year simulation	$\approx$ 2 days	$\approx$ 3.6 days	$\approx$ 11.2 days	$\approx$ 16 days
1 decade simulation	$\approx$ 19 days	$\approx$ 36 days	$\approx$ 112 days	$\approx$ 152 days

Table 6: Sustained performance on the cray XE6 system at CSCS. Note that we use a different number of cores to reach the sustained performance in each of the cases.

4.3 Summary of performance with passive tracers

There is an increasing interest in running the model with a number of passive tracers. Currently, there are two bio-geo-chemical coupler models in the trunk version of the code (DMI-ergom and DMU-ergom) and it thus seems appropriate to evaluate the current performance with passive tracers too. We will confine ourselves to look at 9 (the number of passive tracers used in the DMI-ergom coupler) and 25 (the best estimate of an upper bound on the number of passive tracers in the foreseeable future).

Table 7 summarizes sustained performance on our local XT5 with the different cases running with 9 passive tracers. Note that the ratio $\frac{I_{rv}(9)}{I_{rv}(0)}$ for the variant0 testcase is 2.3 and that this ratio coincides pretty well with the sustained timing ratio 25/11 which is 2.3 too. We do not have sufficient memory to run variant0 with 25 tracers on our local system with more than

11 tasks so we cannot give proper numbers for that case on our local system. Table 8 summarizes sustained performance on the system at CSCS. Note that we can indeed run variant0 with 25 tracers on that system but again we run into memory issues when exceeding 22 tasks (704 AMD Interlagos cores) and it still scales at these core counts so fixing the memory issues will probably lead to better numbers.

XT5, DMI	**MyO v2.1 ergom**	**variant0_9**
24 hour forecast	$<$ 16 minutes	$<$ 25 minutes
10 days forecast	2.4 hours	$<$ 4.1 hours
1 year simulation	$\approx$ 4 days	$\approx$ 7 days
1 decade simulation	$\approx$ 41 days	$\approx$ 65 days

Table 7: Sustained performance on our local cray XT5 system with 9 passive tracers. The ergom model is used in the MyO v2.1 case whereas it is the benchmark coupler that we use in the variant0 case.

XE6, CSCS	**MyO v2.1 ergom**	**variant0_9**	**variant0_25**
24 hour forecast	$<$ 11 minutes	$<$ 13 minutes	$<$ 26 minutes
10 days forecast	1.7 hours	$<$ 130 minutes	$<$ 153 minutes
1 year simulation	$\approx$ 2.6 days	$\approx$ 3.3 days	$\approx$ 6.4 days
1 decade simulation	$\approx$ 26 days	$\approx$ 33 days	$\approx$ 64 days

Table 8: Sustained performance on the cray XE6 system at CSCS with passive tracers. The ergom model is used in the MyO v2.1 case whereas it is the benchmark coupler that we use in the variant0 case. Note that we run the variant0 case with 9 and 25 passive tracers, respectively.

5 MyO V2.1

Table 9 summarizes this setup and figure 2 shows how the four sub-domains nest to each other. The I_r number for the whole setup is 14.3. There are lots of interesting details on this case in [3] where we documented the technical and numerical performance based on physics variables from short model runs. Here we confine ourselves to describe short simulation findings with the passive ergom tracers and finding for the physics parameters from a long simulation, i.e. items that were not covered in [3]. We will also present *as is* findings on the current performance of this testcase.

	IDW	**BS**	**WS**	**NS**
resolution [n.m.]	0.5	3.0	1.0	3.0
mmx [N/S]	482	248	149	348
nmx [W/E]	396	189	156	194
kmx	75	109	24	50
gridpoints	14315400	5109048	557856	3375600
iw2	80904	13056	11581	18908
iw3	1583550	671985	103441	479081
f_{iw3}	11.1%	13.1%	18.5%	14.1%
φ [latitude]	57 35 45N	65 52 30N	55 41 30N	65 52 30N
λ [longitude]	09 20 25E	14 37 30E	06 10 50E	4 07 30W
$\Delta\varphi$	0 0 30	0 3 0	0 1 00	0 3 0
$\Delta\lambda$	0 0 50	0 5 0	0 1 40	0 5 0
dt [sec]	15	30	30	30
maxdepth [m]	78.00	394.40	53.60	696.25
min Δx	827.62	3787.40	1740.97	3787.40
CFL	0.948	0.725	0.751	0.957
I_r	10.0	2.1	0.6	1.5

Table 9: MyO operational setup, version V2.1. The total I_r number for the setup is 14.3.

The documentation of the MyOV2.1 model setup is structured in the following way: First, in section 5.1 we describe the things that one can learn immediately from the series of short simulations using different compilers and compiler options. Then, still with the short simulations, we go into more details and describe a pointwise comparison (section 5.2), a study of compiler issue (section 5.3), and the perfomance of the test case (section 5.4).

After that, we look at a longer simulation (approx. five and half years) in section 5.5, and also here we go details and study specific issues related to water level predictions: We look at the water levels in the Baltic Sea (section 5.6), around Denmark (section 5.7), and in Øresund (section 5.8). It is our hope that we through these sections have given enough insight into the kind of analysis we believe is necessary to do before operationalizing a model.

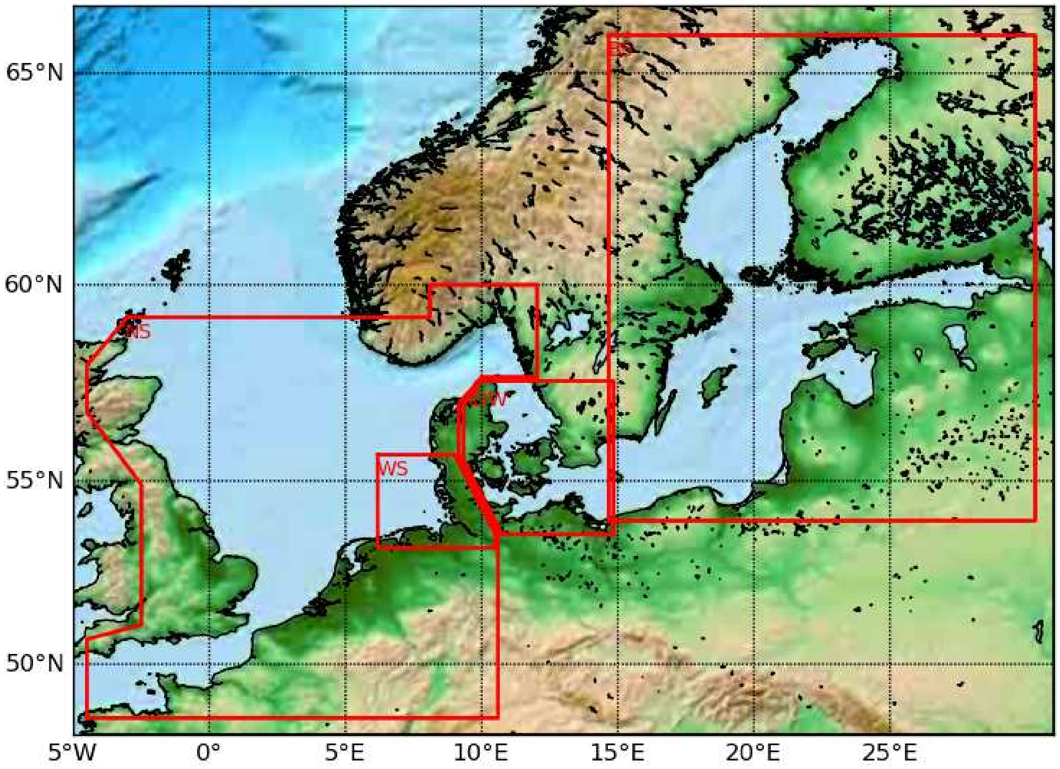


Figure 2: Nesting of the four domains in the MyO V2.1 case and the variant0 case. The difference between the two cases is the resolution in the Baltic Sea domain (BS) which is 3 n.m. and 1 n.m., respectively.

5.1 Short simulations

We have conducted a number of short simulations with different compilers and different compiler flags, cf. table 2 and table 3. We have cross-compared the results stemming from these simulations in order to study how sensible the code and the current test-case is to the choice of compiler, compiler flags and platform. Thus, we attempt to bound the differences in the results emerging from different compilers, i.e. to determine the worst-case differences $\varepsilon_s(f)$ we see across all the runs for each statistical fingerprint s (*avg*, *minimum*, *maximum*) and each prognostic ergom variable f. Let $\varepsilon_s(f)$ and $\delta_s(f)$ be defined by:

$$\varepsilon_s(f) = \max_{c_1,c_2} |s_{c_1}(f) - s_{c_2}(f)|$$

$$\delta_s(f) = \frac{\max_{c_1,c_2} |s_{c_1}(f) - s_{c_2}(f)|}{\max(|s_{c_1}(f)|, |s_{c_2}(f)|)}$$

with $s_{c_1}(p)$ and $s_{c_1}(p)$ being the statistics s for parameter f obtained by compiler c_1 and c_2 respectively. We have shown the bounds obtained for the pure IEEE builds in table 10 and for both IEEE and TUNE builds in table 12. Note that all these results are serial runs but as we will see later the parallel results are binary identical to the serial results so it is sufficient to study the outcome from serial runs.

Browsing these tables it seems that two numbers stick out, namely $\min(T_1)$ in WS and $\max(T_2)$ in IDW, and that calls for a more careful analysis. In tabel 11 we have listed the actual values obtained with each compiler, and from those numbers it might be suggested to exclude a couple of outliers: If we do not include the result from intel for $\max(T_2)$ in IDW and the results from pgi and gfortran for $\min(T_1)$ in WS, we see that the ε drops by a factor of more than 14 in magnitude. The resulting ε values are still high compared to the corresponding values in the other areas so in this case we cannot simply blame the relatively high ε alone on certain single compilers' ability to do numerics: There is, unfortunately, no shortcut here, the implementation of the equations in the source code must be reviewed if the issue should be properly understood and perhaps even dealt with. Whether or not the above findings are related to the physics, e.g. to a the large pointwise difference for salinity in WS during flooding/drying, see e.g. figure 19 in [3], is yet to be investigated.

	NS (ε/δ)	**IDW** (ε/δ)	**WS** (ε/δ)	**BS** (ε/δ)
avg benthos	1.29e-07 / 3.51e-09	2.51e-07 / 8.84e-09	5.80e-07 / 1.04e-08	5.16e-07 / 1.48e-08
min benthos	4.00e-10 / 9.60e-09	0.00e+00 / 0.00e+00	4.40e-09 / 1.46e-09	3.92e-08 / 1.67e-08
max benthos	5.68e-07 / 1.26e-09	3.70e-09 / 1.12e-12	0.00e+00 / 0.00e+00	3.82e-08 / 2.52e-11
avg T_1	1.10e-07 / 2.03e-06	4.83e-08 / 1.46e-06	2.54e-07 / 5.47e-06	1.31e-07 / 4.26e-06
min T_1	0.00e+00 / 0.00e+00	2.28e-07 / 2.54e-05	1.18e-04 / 6.52e-03	5.00e-10 / 1.06e-06
max T_1	2.20e-07 / 1.09e-07	4.47e-07 / 5.58e-08	0.00e+00 / 0.00e+00	7.76e-06 / 3.99e-07
avg T_2	1.42e-07 / 1.14e-07	2.23e-06 / 4.24e-06	5.01e-07 / 6.44e-06	8.13e-07 / 9.28e-07
min T_2	6.00e-10 / 3.03e-08	4.22e-08 / 2.34e-06	2.56e-08 / 1.90e-06	0.00e+00 / 0.00e+00
max T_2	1.11e-06 / 1.16e-07	8.95e-04 / 1.25e-05	0.00e+00 / 0.00e+00	1.19e-07 / 3.09e-09
avg T_3	4.13e-08 / 4.67e-07	5.67e-08 / 8.67e-07	1.75e-07 / 5.99e-06	8.19e-08 / 5.62e-07
min T_3	1.51e-08 / 5.23e-06	4.50e-09 / 1.68e-06	7.00e-10 / 1.34e-07	1.00e-10 / 3.33e-08
max T_3	3.00e-09 / 4.63e-09	1.93e-07 / 8.92e-08	0.00e+00 / 0.00e+00	9.60e-07 / 3.22e-07
avg T_4	5.57e-08 / 9.16e-07	1.81e-08 / 2.27e-06	2.41e-08 / 1.21e-06	1.29e-07 / 4.72e-06
min T_4	0.00e+00 / 0.00e+00	1.35e-08 / 1.51e-04	3.00e-10 / 2.19e-07	0.00e+00 / 0.00e+00
max T_4	3.39e-08 / 3.74e-09	3.83e-06 / 4.43e-07	0.00e+00 / 0.00e+00	9.30e-06 / 8.58e-07
avg T_5	6.74e-08 / 4.50e-07	3.06e-07 / 1.40e-06	3.46e-07 / 1.06e-06	3.24e-07 / 3.42e-06
min T_5	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
max T_5	6.94e-06 / 2.09e-06	1.54e-06 / 3.57e-07	2.31e-06 / 2.65e-07	4.80e-06 / 3.78e-07
avg T_6	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
min T_6	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
max T_6	5.60e-09 / 7.70e-07	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
avg T_7	2.20e-08 / 4.63e-07	9.92e-08 / 3.02e-06	5.08e-08 / 7.64e-07	3.07e-08 / 1.70e-06
min T_7	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
max T_7	2.00e-10 / 2.55e-10	6.60e-09 / 8.41e-09	0.00e+00 / 0.00e+00	4.37e-08 / 5.63e-08
avg T_8	5.77e-08 / 2.31e-07	1.55e-07 / 1.74e-06	6.56e-07 / 3.50e-06	5.84e-07 / 4.32e-06
min T_8	0.00e+00 / 0.00e+00	2.18e-08 / 4.23e-06	2.63e-08 / 4.27e-06	1.00e-10 / 4.16e-08
max T_8	5.31e-08 / 2.02e-08	7.64e-08 / 3.18e-09	5.33e-08 / 1.94e-08	3.47e-07 / 2.29e-08
avg T_9	1.24e-08 / 1.62e-08	9.67e-08 / 1.09e-07	9.47e-08 / 1.22e-07	4.39e-07 / 4.63e-07
min T_9	4.00e-09 / 1.59e-08	1.82e-08 / 6.26e-05	4.00e-09 / 7.37e-09	3.83e-08 / 3.97e-07
max T_9	0.00e+00 / 0.00e+00	2.26e-07 / 1.68e-07	3.00e-10 / 2.05e-10	5.32e-08 / 4.14e-08

Table 10: Worst case differences on statistics for the eco variables between the serial IEEE runs of the MyO operational setup, version V2.1.

Compiler	**IDW, max T_2**	**WS, min T_1**
pgi	71.3978868949	0.0179272771*
cray	71.3978607591	0.0180448939
intel	71.3969921039*	0.0180375865
gfortran	71.3978561956	0.0179619983*
pathscale	71.3978243987	0.0180366909
ε, raw	8.95e-04	1.18e-04
ε, excluding outlier(s)	6.25e-05	8.20e-06

Table 11: A closer look at max(T_2) in IDW and min(T_1) in WS from table 10. Suggested outliers are indicated by *.

As shown in table 13 the parallel runs give results that are binary identical with the serial results. This is true for all compilers used in this study when we use the IEEE compliant compiler flags. But with TUNE flags, this property does not hold for all five compilers in this test for all combinations of openMP and MPI. The most likely reason for this unfortunate behavior is that we used too aggressive optimization flags with some of the compilers. Thus we need to find a useful combination of optimizations flags in the range between the slow, pure IEEE flags and the faster, but also more fragile tuning flags. This will indeed be possible as we have demonstrated in section 5.3.

On the other hand, if it for some compilers eventually turns out that we cannot reproduce the serial results in parallel incarnations with tuning flags, we must analyse the magnitude of the differences introduced, their locations, etc, and reveal what kind of bug we are facing. Then, the code must be fixed appropriately or, perhaps more likely in this particular case, we need to get in contact with the compiler vendor to have the problem solved.

Another finding we did was that statistics for the nine ergom tracers printed to ascii logfiles for runs with more than one MPI task was off compared to statistics from other model runs despite the fact that the results dumped to binary data files were identical. This turned out to be a flaw in the synchronization of the gathering of eco-model tracer data to the global data structure that stores ergom tracers between updating ergom dynamics and printing global statistics. This flaw, which fortunately did not influence the actual results in other ways than cluttering the printed min/ave/max statistics for the nine ergom tracers in runs with more than one MPI task, has been fixed now.

	NS (ε/δ)	**IDW** (ε/δ)	**WS** (ε/δ)	**BS** (ε/δ)
avg benthos	1.29e-07 / 3.51e-09	3.50e-07 / 1.23e-08	5.80e-07 / 1.04e-08	8.38e-07 / 2.41e-08
min benthos	1.10e-09 / 2.64e-08	0.00e+00 / 0.00e+00	5.80e-09 / 1.92e-09	4.05e-08 / 1.72e-08
max benthos	1.17e-06 / 2.60e-09	4.30e-09 / 1.30e-12	0.00e+00 / 0.00e+00	4.15e-08 / 2.74e-11
avg T_1	1.56e-07 / 2.88e-06	1.05e-07 / 3.17e-06	8.80e-07 / 1.89e-05	1.31e-07 / 4.26e-06
min T_1	0.00e+00 / 0.00e+00	2.28e-07 / 2.54e-05	1.21e-04 / 6.72e-03	5.00e-10 / 1.06e-06
max T_1	2.20e-07 / 1.09e-07	5.91e-07 / 7.38e-08	0.00e+00 / 0.00e+00	1.25e-05 / 6.41e-07
avg T_2	2.42e-07 / 1.94e-07	2.23e-06 / 4.24e-06	9.66e-07 / 1.24e-05	1.29e-06 / 1.47e-06
min T_2	8.00e-10 / 4.04e-08	6.55e-08 / 3.64e-06	4.10e-08 / 3.04e-06	0.00e+00 / 0.00e+00
max T_2	1.99e-06 / 2.07e-07	1.47e-03 / 2.06e-05	0.00e+00 / 0.00e+00	3.32e-07 / 8.59e-09
avg T_3	4.32e-08 / 4.88e-07	6.38e-08 / 9.76e-07	2.65e-07 / 9.06e-06	1.21e-07 / 8.33e-07
min T_3	2.02e-08 / 7.00e-06	1.03e-08 / 3.84e-06	7.00e-10 / 1.34e-07	1.00e-10 / 3.33e-08
max T_3	5.10e-09 / 7.88e-09	2.84e-07 / 1.31e-07	0.00e+00 / 0.00e+00	1.52e-06 / 5.10e-07
avg T_4	1.87e-07 / 3.08e-06	2.27e-08 / 2.84e-06	5.02e-08 / 2.51e-06	1.29e-07 / 4.72e-06
min T_4	0.00e+00 / 0.00e+00	1.48e-08 / 1.66e-04	3.00e-10 / 2.19e-07	0.00e+00 / 0.00e+00
max T_4	7.42e-08 / 8.19e-09	7.45e-06 / 8.62e-07	0.00e+00 / 0.00e+00	1.41e-05 / 1.30e-06
avg T_5	6.74e-08 / 4.50e-07	3.06e-07 / 1.40e-06	1.03e-06 / 3.16e-06	4.79e-07 / 5.05e-06
min T_5	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
max T_5	8.27e-06 / 2.50e-06	3.25e-06 / 7.53e-07	3.39e-06 / 3.90e-07	4.80e-06 / 3.78e-07
avg T_6	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
min T_6	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
max T_6	1.13e-08 / 1.55e-06	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
avg T_7	3.55e-08 / 7.46e-07	1.17e-07 / 3.57e-06	1.18e-07 / 1.78e-06	3.52e-08 / 1.95e-06
min T_7	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
max T_7	3.00e-10 / 3.83e-10	1.31e-08 / 1.67e-08	0.00e+00 / 0.00e+00	6.24e-08 / 8.04e-08
avg T_8	1.29e-07 / 5.18e-07	1.65e-07 / 1.86e-06	1.39e-06 / 7.42e-06	5.84e-07 / 4.32e-06
min T_8	0.00e+00 / 0.00e+00	2.40e-08 / 4.66e-06	4.18e-08 / 6.78e-06	1.00e-10 / 4.16e-08
max T_8	5.31e-08 / 2.02e-08	1.68e-07 / 7.01e-09	5.33e-08 / 1.94e-08	4.84e-07 / 3.19e-08
avg T_9	1.98e-08 / 2.58e-08	1.43e-07 / 1.61e-07	1.36e-07 / 1.75e-07	4.39e-07 / 4.63e-07
min T_9	6.50e-09 / 2.59e-08	2.75e-08 / 9.46e-05	4.30e-09 / 7.92e-09	4.92e-08 / 5.09e-07
max T_9	1.00e-10 / 1.01e-10	4.01e-07 / 2.98e-07	3.00e-10 / 2.05e-10	1.40e-07 / 1.09e-07

Table 12: Worst case differences on statistics for the eco variables within the pool of serial IEEE+TUNE runs of the MyO operational setup, version V2.1.

md5sum	6 hours simulation files
b1917b977094f360786d8ed3863b9775	pathscale/0_0_dmiergom_ieee/biodat.00
b1917b977094f360786d8ed3863b9775	pathscale/0_12_dmiergom_openmp_ieee/biodat.00
b1917b977094f360786d8ed3863b9775	pathscale/8_0_dmiergom_mpi_ieee/biodat.00
b1917b977094f360786d8ed3863b9775	pathscale/8_12_dmiergom_mpi_openmp_ieee/biodat.00
e9bbcf34b10db8744ccbec4f20e7653f	pathscale/0_0_dmiergom_ieee/tempdat.00
e9bbcf34b10db8744ccbec4f20e7653f	pathscale/0_12_dmiergom_openmp_ieee/tempdat.00
e9bbcf34b10db8744ccbec4f20e7653f	pathscale/8_0_dmiergom_mpi_ieee/tempdat.00
e9bbcf34b10db8744ccbec4f20e7653f	pathscale/8_12_dmiergom_mpi_openmp_ieee/tempdat.00
0bc0884e335800e242a1e610f9d61270	pathscale/0_0_dmiergom_ieee/bio_restart.00
0bc0884e335800e242a1e610f9d61270	pathscale/0_12_dmiergom_openmp_ieee/bio_restart.00
0bc0884e335800e242a1e610f9d61270	pathscale/8_0_dmiergom_mpi_ieee/bio_restart.00
0bc0884e335800e242a1e610f9d61270	pathscale/8_12_dmiergom_mpi_openmp_ieee/bio_restart.00
2d64547e019eeae8bfedb2109328dbeb	pathscale/0_0_dmiergom_ieee/restart.00
2d64547e019eeae8bfedb2109328dbeb	pathscale/0_12_dmiergom_openmp_ieee/restart.00
2d64547e019eeae8bfedb2109328dbeb	pathscale/8_0_dmiergom_mpi_ieee/restart.00
2d64547e019eeae8bfedb2109328dbeb	pathscale/8_12_dmiergom_mpi_openmp_ieee/restart.00

Table 13: Comparison of 4 configure options (default, `--enable-openmp`, `--enable-mpi`, `--enable-openmp --enable-mpi`) all using the IEEE compiler flag. That is, IEEE-serial (0_0_ieee), IEEE-openMP (0_12_openmp_ieee), IEEE-MPI (8_0_mpi_ieee) and IEEE-openMP-MPI (8_12_openmp_mpi_ieee). Note that we get binary identical results across the 4 configure options when using the IEEE flag category.

5.2 Pointwise cross-comparisons

Analysing the pointwise differences between two compilers can tell us a lot about the uncertainty of the model result, but we need to be very careful in judging the origin of these pointwise differences, i.e. whether they are results of an unsafe implementation of the model code or if certain compilers have some issue (e.g. with their implementation of elementary functions).

As an example[11], it was found that the largest discrepancy across compilers occurred in the subsurface water, around the depths of the thermocline, c.f. figure 3. Tracer 9 (dissolved oxygen) differes a couple of percent relative to its absolute variability. The figure clearly indicates that the largest differences occur between 5-20 m depths while the difference in the surface and bottom layers are very small. The different results across compilers in the biological variables could be traced to a similar pattern in the temperature, c.f. figure 3 where we see a difference of 0.3 °C between runs using two different compilers and everything else unchanged.

It should be noted that a similar behavior as we see in figure 3 for temperature is not observed for salinity; it should therefore be safe to rule out the tracer advection and the tracer diffusion schemes as possible main sources of the issue since these are the same used for all tracers tempeature, salinity and eco-model tracers. Then we are left with two more or less obvious suspects: the thermodynamics and the calculation of the diffusivity for temperature (which is different from that of salinity, see [2]). Another possibility is that input parameters can be interpreted slightly differently by different compilers. Recent studies revealed that we still see the issues from figure 3 when we make a test run without the thermodynamics module activated. At the time of writing we can unfortunately not say anything more specific as to the cause of the issue; whether it should be found solely in the calculation of the diffusivity for temperature or it arises from a combination from multiple source is yet to be investigated.

[11] Thanks to Lars Jonasson, DMI, for sharing his findings.

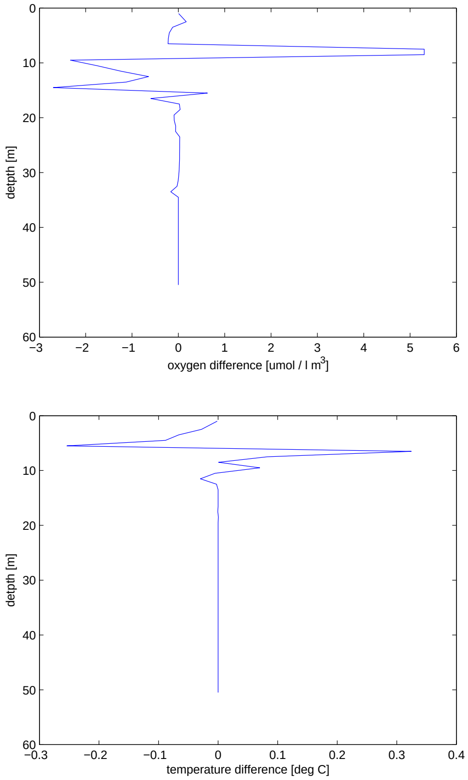


Figure 3: Profiles of the difference between gfortran and cray for the biological tracer 9 (dissolved oxygen) and temperatures. The two profiles are not taken at the same geographical location but are selected to show the largest difference.

5.3 Study of optimization issues

As noted above, we did not get results from parallel runs which were binary identical to the serial results when we used tuning flags for all compilers, but only for the IEEE runs. We claimed, however, that we would be able to have that necessary property for the TUNE runs too, simply be finding and using a suitable combination of optimization flags.

Below we describe how we approached the problem for the pgi compiler and for the cray compiler. We will use our findings for these two compilers in the future, and if time permits and applications demand we will also study and fix the behavior for other compilers but this is out of the scope of the present report.

The pgi compiler:
First, we need to understand why the PGI compilers gives different MPI parallel results when using the TUNE flag. Moreover, we need to understand why we do not see this issue on the XE6 system at CSCS.

The observant reader may notice that there is a slight deviation in the compiler flags chosen on the two systems and a plausible explanation could thus be that the issue is due to this. We tried different optimization flags and ran 20 incarnations $(1, \ldots, 20$ MPI tasks) for each flag, cf. table 14. After this finding, we reproduced the issue on the XE6 system at CSCS too.

Flags	**Result**	**1 node**	**20 nodes**
-O2	binary identical	1242 sec	256 sec
-O2 -Mipa=fast,inline	binary identical	1218 sec	257 sec
-O3	binary identical	1239 sec	257 sec
-O3 -Mipa=fast,inline	binary identical	1215 sec	258 sec
-fastsse -Mipa=fast,inline	not binary identical	1207 sec	250 sec
-fastsse	not binary identical	1201 sec	249 sec

Table 14: List of compilers flags for the 11.10.0 release of the PGI compiler and reported behaviour. Note that when we see binary identical results the results are the same as with the other flags, e.g. `-O2` and `-O3 -Mipa=fast,inline` produces binary identical results and so forth.

We were curious to see how large the deviations among the 20 runs actually were. Thus, we will cross-compare statistics emerging in the 20 logfiles and we will cross-compare the pointwise differences emerging in the 20 `tempdat` files. As for the pointwise study, we will confine ourselves to study the differences at the surface ($L = 1$) and at the bottom (the bottom layer is attained at different levels $L \in 1, \ldots L_{\max}$ depending on the point at hand and by definition we let $L = 0$ be the bottom layer). Let the pointwise deviation $\varepsilon_p(f, L)$ for field f at layer L be defined by:

$$\varepsilon_p(f, L) = \max_{iw \in L, c_1, c_2} |f_{c_1}(iw) - f_{c_2}(iw)|$$

with L here being either the surface layer or the bottom layer. We suspect that differences caused by the `-fastsse` flag are due to vectorization of reduction loops. When the MPI ranks change, the lengths of arrays differs and so they have different "remainder" loops in vector mode.

Table 15 shows the worst-case pointwise differences whereas table 16 and table 17 shows the worst-case statistical differences.

	NS (ε/δ)	**IDW** (ε/δ)	**WS** (ε/δ)	**BS** (ε/δ)
$\varepsilon_p(zlev, 1)$	3.39e-04/1.40e-04	9.70e-04 /1.63e-03	1.56e-04 /8.03e-05	8.91e-03/1.06e-02
$\varepsilon_p(salt, 0)$	4.35e-02/1.23e-03	6.46e-01 /1.85e-02	6.86e-03 /1.96e-04	8.76e-02/2.50e-03
$\varepsilon_p(salt, 1)$	1.41e-02/3.97e-04	2.16e-01 /6.16e-03	4.85e-03 /1.38e-04	9.53e-02/2.72e-03
$\varepsilon_p(temp, 0)$	5.97e-02/4.34e-03	5.01e-01 /3.63e-02	6.71e-03 /4.63e-04	9.26e-01/7.37e-02
$\varepsilon_p(temp, 1)$	4.88e-03/3.55e-04	8.24e-02 /6.08e-03	4.50e-03 /3.11e-04	4.90e-01/3.90e-02

Table 15: Worst case pointwise differences when comparing the 20 TUNE results emerging from the PGI compiler.

The cray compiler:
First, we notice that the cray binary runs pretty fast with the slow IEEE flags, actually just as fast or faster than other compilers can do with their aggressive optimization flags. So, it is really a luxury problem we are facing with cray. Anyway, we will like to know the reason and to use some kind of tuning to improve the execution speed.

As seen from table 3 we used two optimization flags with the cray compiler on our local XT5, namely the default level 2 flag (-O2) for moderate optimization, combined with the -ipa5 flag for aggressive interprocedural analysis (IPA). For the IEEE builds we used conservative optimization (-O1) together with the flag for maximum safety of floating-point optimizations (-Ofp0), cf. table 2. Thus, one would expect to be able to find a compromise somewhere in the range laid out by these two sets of flags.

One thing to notice is that when nothing else is specified the cray compiler will use the default setting for moderate floating-point optimization (-Ofp2). Thus, explicitly requiring the safe -Ofp0 flag, we played with combinations of the other involved flags (-O1 or -O2, and different -ipa$N$ with $N = 0, \ldots, 5$) and found that with the flags

-Ofp0 -O1 -ipa5

	NS (ε/δ)	**IDW** (ε/δ)	**WS** (ε/δ)	**BS** (ε/δ)
Avg salinity	2.43e-07 / 6.98e-09	6.30e-06 / 4.16e-07	3.92e-06 / 1.15e-07	1.56e-06 / 2.51e-07
RMS for salinity	2.14e-07 / 6.15e-09	8.00e-06 / 4.59e-07	3.95e-06 / 1.16e-07	1.36e-06 / 2.10e-07
STD for salinity	3.29e-06 / 2.63e-06	5.46e-06 / 6.33e-07	8.14e-07 / 4.25e-07	1.55e-06 / 8.88e-07
Avg temp	2.37e-07 / 2.98e-08	2.64e-06 / 3.29e-07	2.22e-06 / 2.52e-07	1.54e-05 / 3.53e-06
RMS for temp	2.57e-07 / 3.19e-08	3.47e-06 / 4.21e-07	2.14e-06 / 2.43e-07	1.00e-05 / 2.16e-06
STD for temp	2.68e-07 / 2.20e-07	3.91e-06 / 2.05e-06	3.07e-06 / 3.64e-06	3.75e-05 / 2.32e-05
Min salinity	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
Max salinity	0.00e+00 / 0.00e+00	1.06e-06 / 3.05e-08	4.13e-09 / 1.18e-10	1.32e-08 / 9.66e-10
Min temp	3.45e-08 / 6.45e-09	8.73e-07 / 4.21e-07	1.46e-07 / 2.20e-08	2.61e-07 / 1.91e-06
Max temp	0.00e+00 / 0.00e+00	3.74e-08 / 2.71e-09	0.00e+00 / 0.00e+00	7.52e-06 / 5.98e-07
Min u	0.00e+00 / 0.00e+00	2.94e-04 / 3.91e-04	0.00e+00 / 0.00e+00	5.06e-05 / 1.20e-04
Max u	3.82e-06 / 2.22e-06	2.35e-04 / 2.76e-04	3.56e-06 / 2.37e-06	6.22e-03 / 8.22e-03
Min v	1.17e-05 / 1.04e-05	1.18e-04 / 1.56e-04	1.60e-12 / 1.28e-12	2.50e-05 / 6.22e-05
Max v	0.00e+00 / 0.00e+00	1.26e-04 / 1.09e-04	1.78e-06 / 1.25e-06	5.96e-03 / 5.42e-03
Avg z	9.75e-07 / 9.44e-06	2.80e-06 / 1.74e-05	1.59e-06 / 2.70e-06	6.84e-07 / 2.11e-06
RMS for z	1.02e-06 / 1.43e-06	3.99e-06 / 1.83e-05	7.60e-07 / 1.08e-06	4.21e-06 / 1.19e-05
STD for z	8.84e-07 / 1.26e-06	2.95e-06 / 2.01e-05	1.29e-06 / 3.38e-06	8.99e-06 / 6.35e-05
Min z	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	1.53e-08 / 7.47e-08	2.88e-05 / 4.97e-04
Max z	0.00e+00 / 0.00e+00	1.21e-04 / 2.25e-04	0.00e+00 / 0.00e+00	1.95e-05 / 2.29e-05

Table 16: Worst case differences on statistics between the 20 PGI runs of the MyO operational setup, version V2.1.

	NS (ε/δ)	**IDW** (ε/δ)	**WS** (ε/δ)	**BS** (ε/δ)
avg benthos	2.92e-08 / 7.95e-10	1.65e-07 / 5.80e-09	5.66e-08 / 1.02e-09	1.20e-06 / 3.46e-08
min benthos	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	1.00e-10 / 3.31e-11	6.45e-08 / 2.75e-08
max benthos	4.26e-07 / 9.45e-10	1.80e-09 / 5.46e-13	0.00e+00 / 0.00e+00	2.71e-08 / 1.79e-11
avg T_1	6.12e-08 / 1.13e-06	1.14e-07 / 3.42e-06	2.80e-07 / 6.02e-06	9.65e-08 / 3.13e-06
min T_1	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	8.73e-06 / 4.86e-04	4.00e-10 / 8.50e-07
max T_1	0.00e+00 / 0.00e+00	3.95e-07 / 4.93e-08	0.00e+00 / 0.00e+00	6.02e-06 / 3.09e-07
avg T_2	6.42e-08 / 5.16e-08	2.49e-06 / 4.74e-06	3.29e-07 / 4.23e-06	1.12e-06 / 1.28e-06
min T_2	1.00e-10 / 5.05e-09	4.00e-10 / 2.22e-08	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
max T_2	5.79e-07 / 6.02e-08	1.67e-03 / 2.34e-05	0.00e+00 / 0.00e+00	2.77e-07 / 7.19e-09
avg T_3	1.33e-08 / 1.50e-07	4.19e-08 / 6.41e-07	1.82e-07 / 6.24e-06	1.27e-07 / 8.74e-07
min T_3	1.90e-09 / 6.58e-07	6.70e-09 / 2.50e-06	1.00e-10 / 1.91e-08	1.00e-10 / 3.33e-08
max T_3	2.30e-09 / 3.55e-09	2.47e-07 / 1.14e-07	0.00e+00 / 0.00e+00	7.74e-07 / 2.59e-07
avg T_4	1.29e-08 / 2.12e-07	2.09e-08 / 2.62e-06	5.00e-09 / 2.50e-07	1.35e-07 / 4.94e-06
min T_4	0.00e+00 / 0.00e+00	1.37e-08 / 1.54e-04	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
max T_4	0.00e+00 / 0.00e+00	5.77e-06 / 6.68e-07	0.00e+00 / 0.00e+00	9.10e-06 / 8.40e-07
avg T_5	2.22e-08 / 1.48e-07	3.62e-07 / 1.66e-06	1.96e-07 / 6.03e-07	4.03e-07 / 4.25e-06
min T_5	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
max T_5	0.00e+00 / 0.00e+00	2.57e-06 / 5.95e-07	0.00e+00 / 0.00e+00	4.08e-06 / 3.22e-07
avg T_6	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
min T_6	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
max T_6	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
avg T_7	5.20e-09 / 1.09e-07	1.24e-07 / 3.78e-06	5.70e-08 / 8.57e-07	3.83e-08 / 2.12e-06
min T_7	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00	0.00e+00 / 0.00e+00
max T_7	0.00e+00 / 0.00e+00	1.66e-08 / 2.12e-08	0.00e+00 / 0.00e+00	1.20e-07 / 1.54e-07
avg T_8	4.50e-08 / 1.80e-07	1.47e-07 / 1.65e-06	3.74e-07 / 1.99e-06	6.24e-07 / 4.61e-06
min T_8	0.00e+00 / 0.00e+00	1.36e-08 / 2.64e-06	0.00e+00 / 0.00e+00	1.00e-10 / 4.16e-08
max T_8	0.00e+00 / 0.00e+00	1.15e-07 / 4.80e-09	0.00e+00 / 0.00e+00	4.29e-07 / 2.83e-08
avg T_9	8.40e-09 / 1.10e-08	1.15e-07 / 1.30e-07	2.94e-08 / 3.77e-08	5.02e-07 / 5.30e-07
min T_9	2.90e-09 / 1.15e-08	2.05e-08 / 7.05e-05	0.00e+00 / 0.00e+00	5.97e-08 / 6.18e-07
max T_9	0.00e+00 / 0.00e+00	2.54e-07 / 1.88e-07	0.00e+00 / 0.00e+00	1.66e-07 / 1.29e-07

Table 17: Worst case differences on statistics for the eco variables between the 20 PGI runs of the MyO operational setup, version V2.1.

we are able to obtain parallel results which are identical to the serial results. With these flags, it runs slightly faster than with the pure IEEE flags of table 2 and slower than TUNE flags of table 3, but still fast enough to outperform most of the other compilers with TUNE flags, only surpassed by pgi with the above-mentioned flags.

5.4 Performance

First, we have studied the serial performance of the individual time steps doing simple serial timings from within the source code. Looking at the timing of the first 120 (out of the 720) individual time steps, we see that they fall into five categories in this testcase, cf. figure 4. A default workload (720 times) stemming from the basic physics model, an additional workload from tracer advection and diffusion, called tflow, (360 times) on the even time steps and an additional turbulence workload (360 times) on the odd time steps. Then we have an additional ergom load (180 times) every fourth time step and finally an additional workload due to IO (tempdat output and forcing input for every 120 time steps). The timings of these categories are summarized in table 18.

	pgi	**cray**	**pathscale**	**gnu**	**intel**
	[sec]	[sec]	[sec]	[sec]	[sec]
d+tflow part	10.9	11.1	12.8	11.7	11.7
d+turbulence part	3.3	3.2	3.7	3.2	3.4
ergom part	1.6	1.6	2.3	2.0	1.1
IO part	2.8	2.1	2.5	3.2	2.1
Total time	5405	5472	6362	5891	5663

Table 18: Serial timing of each category using different compilers. The values above are average values. Note that one has to sum the column values to get the average timing for the time step that involves the chosen columns.

Second, we have studied the parallel performance. The overall scaling of the setup with the ergom tracers is shown for different compilers in figure 5 and figure 6.

Finally, we had a look at the initialization prior to the timeloop and also zoomed in on the IO activities comparing the timings emerging from different compilers cf. table 19. Note that *IO bathymetry* is part of the *Initial-*

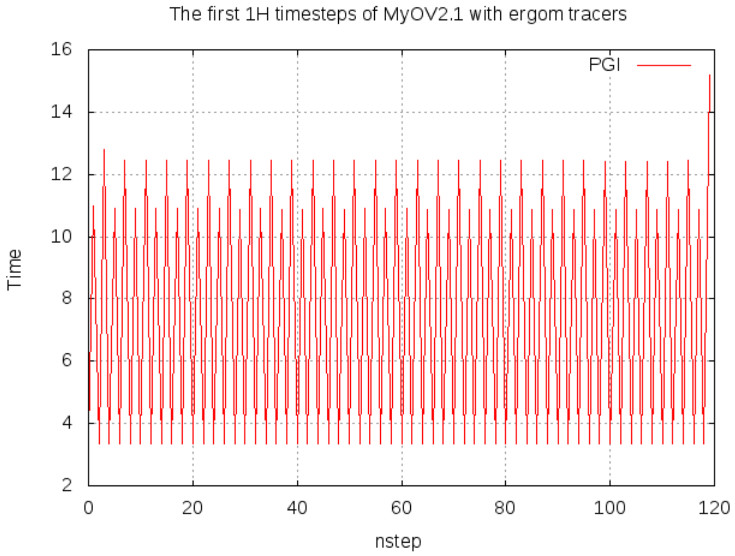


Figure 4: The individual time steps for the PGI compiler, serial tune

isation time and it should be mentioned that it is trivial to improve this. One can simply add `bathy_asc2bin=.true.` or `bathy_asc2mpibin=.true.` into the namelist `cfglist` in the configuration `cfg.nml` to convert the ascii input files to binary fortran files or pure binary files in the MPI-IO sense. One of these files can then be used instead of the ascii file (by specifying `bathy_bin=.true.` or `bathy_mpibin=.true.`, respectively to improve the performance. We have included the numbers for binary bathymetry input in table 19.

	pgi	**cray**	**pathscale**	**gnu**	**intel**
Initialisation	15.2/13.7	14.4	15.0/12.5	19.8/14.3	13.4/11.5
IO bathymetry	4.7/1.9	4.6/0.8	3.3/0.8	7.1/1.5	3.9/1.0
IO tempdat	5.0	4.0	5.5	6.4	4.3
IO restart	0.98	1.9	1.1	1.2	1.5

Table 19: Timings of the initialisation and from the most expensive IO parts using different compilers.

We cannot state anything on performance without looking at some profiles

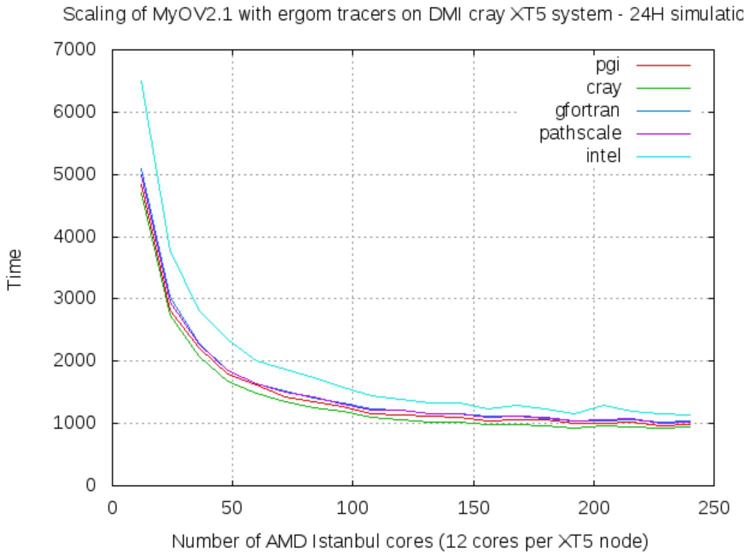


Figure 5: Scaling of ergom using different compilers. Note that the 20 results (number of mpi tasks $\in \{1,\ldots,20\}$) from all compilers but the PGI compiler above generate binary identical results (`tempdat`, `restart`, `biodat`, `bio_restart`).

so we have gathered a few snapshots of profiles, cf. figure 7, 8, and 9. All the profiles shown are done using PGI generated binaries.

5.5 Longer simulation

We have also conducted a longer simulation (August 2006 - December 2011, both months included) of this case but without the extra set of passive ergom tracers. Simple statistics on the physical variables from this study is shown in table 20.

One thing is to look at the attained ranges of different parameters from e.g. table 20 and explain them from oceanographic knowledge of the domains, i.e. explain that they fall within the expected ranges. Another thing is to look at and understand the time evolution of the statistical fingerprints of the physical parameters over the simulation period; that is to explain trends, signals and extremes. We have plotted and looked at time series

	NS	IDW	WS	BS
u	[-4.63; 4.21]	[-2.19; 2.00]	[-2.41; 2.50]	[-2.02; 2.86]
v	[-3.99; 5.14]	[-3.35; 2.55]	[-2.79; 2.84]	[-2.43; 2.16]
s	[0.00; 35.40]	[0.00; 35.01]	[0.00; 35.12]	[0.00; 18.98]
avg s	[34.58; 34.94]	[14.32; 19.65]	[32.72; 34.76]	[6.33; 6.74]
rms s	[34.61; 34.95]	[16.61; 21.41]	[32.79; 34.79]	[6.56; 7.02]
std s	[0.65; 1.54]	[7.64; 9.84]	[1.34; 2.95]	[1.71; 1.98]
t	[-1.83; 25.69]	[-1.79; 26.67]	[-1.87; 26.80]	[-0.44; 26.56]
avg t	[5.88; 12.24]	[0.66; 17.06]	[3.15; 19.44]	[2.06; 9.82]
rms t	[6.05; 12.97]	[1.72; 18.00]	[3.30; 19.44]	[2.81; 11.45]
std t	[0.88; 4.93]	[0.33; 6.11]	[0.24; 4.38]	[1.41; 6.64]
z	[-6.22; 7.90]	[-1.60; 1.76]	[-3.24; 5.36]	[-1.02; 2.99]
avg z	[-0.54; 0.78]	[-0.37; 1.05]	[-1.48; 2.50]	[-0.19; 1.04]
rms z	[0.24; 1.61]	[0.04; 1.07]	[0.22; 2.61]	[0.02; 1.09]
std z	[0.23; 1.52]	[0.03; 0.84]	[0.17; 0.83]	[0.01; 0.44]

Table 20: Ranges attained for the shown parameters in each domain during a simulation from the 1st of August 2006 until the 31th December 2011. The statistics is collected for each parameter at a 6 hour interval throughout the entire period.

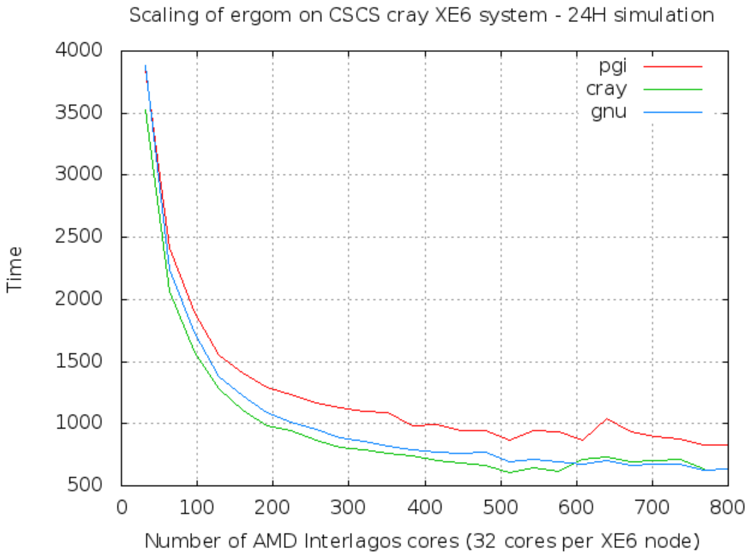


Figure 6: Scaling of ergom using different compilers. For the PGI and the CRAY compiler all the MPI runs generate binary identical results (`tempdat`, `restart`, `biodat`, `bio_restart`). This is not the case for the GNU compiler unless we remove the `-ffast-math` option from the compiler flags.

of the minimum, average and maximum values of temperature, salinity and sea level for all for domains; we found no alarming or unexpected behavior in the first three domains, so they will not be shown here. In the fourth domain, BS, we did, however, find issues worth some more explanation.

Figure 10 shows the minimum, average and maximum value of `z` for the BS domain. This figure and table 20 show that we seem to be on the safe side with a top layer thickness of 2 meter since the modelled `z` never comes near -2 m.

Also from figure 10 (as well as from the previous table 20) we can see that the maximum value of `z` in BS in the model reaches 2.5-3 meter during the storms in October 2006, January 2007 and in December 2011. If this reflects the true behavior in nature or if it is a model artifact is a subject for further investigations, cf. section 5.6.

```
|| Time%  |       Time  |  Calls  |Group
||  56.4% | 3117.380162 |  2880.0 |tflow_tflow_int_
||  14.1% |  777.984000 |  3600.0 |momeqs_wrapping_momeqs_default_
||  11.4% |  632.516147 |  1440.0 |turbmodels_turbmodel_
||   5.9% |  326.496113 |   720.0 |biomod_bio_dynamics_
||   1.7% |   94.366162 |  3600.0 |smagorinsky_smag_
||   1.5% |   83.448810 |  1444.0 |thermodynamic_thermodyn_1_
||   1.5% |   80.559519 |   720.0 |cmod_hydrodynamics_solvehydrodynamics_
||   1.3% |   72.235968 |  1444.0 |cmod_dens_dens_
||   1.1% |   61.041499 |  3600.0 |masseqs_masseqs_solver_z_
||   1.1% |   58.798057 |  3600.0 |smagorinsky_deform_
```

Figure 7: Serial profile of a 6H simulation for myo v2.1 with 9 ergom tracers, PGI, tune flag

```
|| Time%  |       Time  |   Calls  |Group
|| 100.0% | 1278.634534 | 500017.0 |Total
|------------------------------------------------------------------------------
||  94.2% | 1204.080571 | 456709.0 |USER
||-----------------------------------------------------------------------------
||  13.1% |  167.500938 |   1440.0 |tflow_tflow_int_.REGION@li.2471
||  12.1% |  155.043252 |   1440.0 |tflow_tflow_int_.REGION@li.2431
||  11.7% |  149.262115 |   1440.0 |tflow_tflow_int_.REGION@li.2485
||   8.9% |  113.708664 |   1440.0 |tflow_tflow_int_.REGION@li.2498
||   6.6% |   84.453482 |   1440.0 |tflow_tflow_int_.REGION@li.2516
||   6.2% |   78.723789 |   3600.0 |momeqs_wrapping_momeqs_default_
||   5.8% |   74.765771 |   1440.0 |tflow_tflow_int_.REGION@li.2454
||   5.5% |   70.225735 |   1440.0 |tflow_tflow_int_.REGION@li.2602
||   4.8% |   60.915812 |   1440.0 |turbmodels_turbmodel_
||   3.1% |   40.161192 |   4320.0 |dmi_mpi_dmpi_gather_copy_cmp_
||   2.2% |   27.835280 |    720.0 |biomod_bio_dynamics_
||   1.7% |   21.742877 |   3600.0 |cmod_hydrodynamics_solvemasseq_.REGION@li.564
||   1.1% |   14.525062 |   4320.0 |dmi_mpi_dmpi_gather_copy_
||   1.0% |   12.638315 |   3600.0 |smagorinsky_deform_
||   1.0% |   12.163902 |   3600.0 |smagorinsky_smag_
||=============================================================================
||   5.8% |   74.553165 |  43295.0 |OMP
||-----------------------------------------------------------------------------
||   1.7% |   21.991757 |   1440.0 |tflow_tflow_int_.REGION@li.2471(ovhd)
||   1.1% |   13.743574 |    720.0 |coupler_coupler_dynamics_.REGION@li.206(ovhd)
||=============================================================================
|   0.0% |    0.000791 |     11.0 |PTHREAD
|   0.0% |    0.000007 |      2.0 |MPI
```

Figure 8: openmp_mpi, 12 threads, 1 task profile of a 6H simulation for ergom, PGI, tune flag

```
|| Time%  |       Time |   Calls   |Group
|| Time%  |       Time |      Imb. |  Imb.   |    Calls  |Group
||        |            |      Time | Time%   |           | Function
|| 100.0% | 278.363530 |        -- |      -- | 505871.7 |Total
|--------------------------------------------------------------------------------------------
|  97.5% | 271.343673 |        -- |      -- | 459480.4 |USER
||-------------------------------------------------------------------------------------------
||  37.5% | 104.344437 | 21.428275 |  17.9% |  28812.0 |dmi_mpi_dmpi_distribute_halo_nc_
||  11.9% |  33.059488 |  4.767601 |  13.3% |   4320.0 |dmi_mpi_dmpi_gather_copy_cmp_
||  10.8% |  30.023821 | 11.098111 |  28.4% |  41764.0 |dmi_mpi_dmpi_halo_nonblocking_
||   3.9% |  10.992847 |  3.534521 |  25.6% |     88.0 |dmi_mpi_dmpi_gather_all_nc_
||   2.5% |   7.072585 | 10.895363 |  63.8% |   4320.0 |dmi_mpi_dmpi_gather_copy_
||   2.5% |   6.965319 |  0.806633 |  10.9% |   3162.0 |dmi_mpi_dmpi_gather_all_
||   2.2% |   6.181823 |  0.379245 |   6.1% |   1440.0 |tflow_tflow_int_.REGION@li.2485
||   2.1% |   5.919510 |  0.222921 |   3.8% |   1440.0 |tflow_tflow_int_.REGION@li.2431
||   2.0% |   5.696745 |  0.320462 |   5.6% |   1440.0 |tflow_tflow_int_.REGION@li.2471
||   1.7% |   4.662225 |  5.135529 |  55.2% |   1080.0 |dmi_mpi_dmpi_barrier_
||   1.6% |   4.318479 |  0.244307 |   5.6% |   1440.0 |tflow_tflow_int_.REGION@li.2498
||   1.5% |   4.149626 |  0.744381 |  16.0% |      6.0 |dmi_mpi_dmpi_broadcast_met_info_
||   1.4% |   3.803432 |  0.334207 |   8.5% |   1440.0 |tflow_tflow_int_.REGION@li.2516
||   1.3% |   3.574325 |  1.283442 |  27.8% |      4.4 |dmi_mpi_dmpi_scatter_cmp_
||   1.3% |   3.517650 |  0.383785 |  10.4% |   3600.0 |momeqs_wrapping_momeqs_default_
||   1.2% |   3.239443 |  0.269464 |   8.1% |   1440.0 |turbmodels_turbmodel_
||   1.1% |   3.193581 |  0.223689 |   6.9% |   1440.0 |tflow_tflow_int_.REGION@li.2454
||   1.0% |   2.806342 |  0.095372 |   3.5% |      1.0 |exit
||   1.0% |   2.701762 |  0.256606 |   9.1% |   1440.0 |tflow_tflow_int_.REGION@li.2602
||===========================================================================================
|   2.0% |   5.673209 |        -- |      -- |  46378.2 |OMP
||-------------------------------------------------------------------------------------------
|   1.1% |   2.958088 |  0.238067 |   7.8% |      1.0 | dmi_omp_domp_init_.REGION@li.46(ovhd)
||===========================================================================================
|   0.5% |   1.345873 |        -- |      -- |      2.0 |MPI
|   0.0% |   0.000774 |  0.000144 |  16.5% |     11.0 |PTHREAD
```

Figure 9: openmp_mpi, 12 threads, 20 task profile of a 6H simulation for ergom, PGI, tune flag

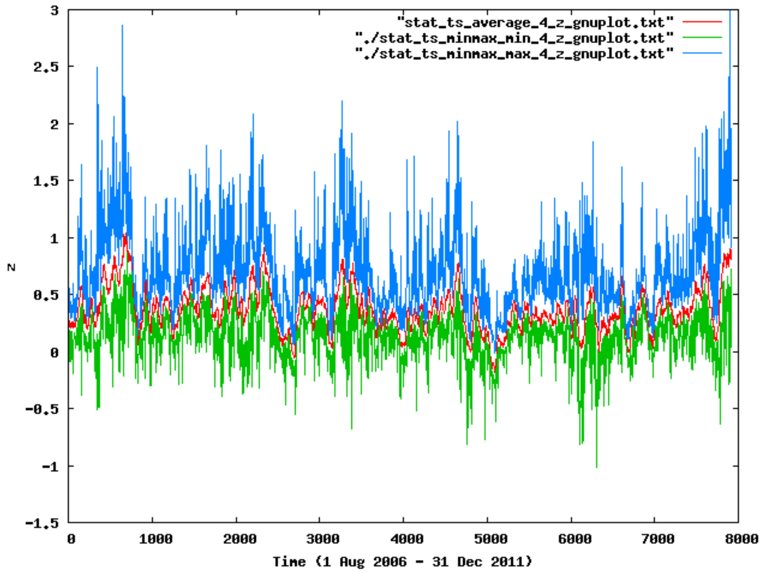


Figure 10: Minimum (green), average (red) and maximum (blue) instantaneous values of water level in the BS domain, plotted every 6 hour.

Figure 11 shows statistical parameters for salinity in the BS domain. During the considered period there is hardly any inflow of dense water to this BS model domain, so the maximum value has a clear decreasing trend as expected due to vertical mixing. Later in the period there are some inflows to the BS model domain of more or less significant magnitude which help maintaining a level above 14 throughout the period. The average salinity is slowly decreasing due to continuous fresh water supply from the rivers. This is very much consistent with observations, cf. [11]: Since the major Baltic inflow in 2003 there has not been added significant new salt to the Baltic Sea, except during some minor events. The baroclinic inflow of August/September 2006 (the beginning of our simulation period) was likely the most significant during the considered period, and as shown in [11] the bottom salinity in the Bornholm Deep decreased from approximately 16.3 in May 2007 to approximately 14.6 in May 2011. Our modelled maximum salinity in BS is consistent with this decrease, though the modelled decrease seems to be slightly too rapid. The reason being, we believe, mainly due to insignificant spatial resolution; preliminary tests have shown that increasing the horizontal resolution to 1 n.m. and improving the vertical resolution as well will help on this problem (this is the variant0 setup, cf. section 6, which will be reported elsewhere).

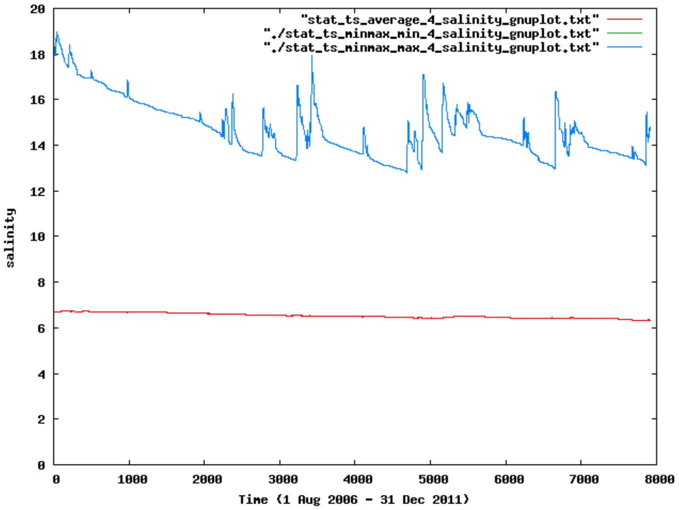


Figure 11: Minimum (green), average (red) and maximum (blue) instantaneous values of salinity in the BS domain, plotted every 6 hour. Please note, the minimum salinity is zero through out the period due to fresh water river inflows, and therefore the green curve is coincident with the zero-line and thus cannot be seen on this plot.

Figure 12 shows minimum, average and maximum temperature in the BS domain with their respective expected annual patterns. One can speculate if the apparent decrease in average temperature (both annual maxima and annual minima of the red curve) is real, if it stems from the model adjusting to the "real word physics" from a start with an initial field that is too much off, if it is related to the meteo forcing, or if this trend is simply just wrong.

5.6 The water level in the Baltic Sea

In this section we will try to explain the fact that the model reaches water levels of 2.5-3 meter in the Baltic Sea during the storms in October 2006, January 2007 and in December 2011. Below we have picked the 5 highest high-water in the simulation in the BS domain from the 6 hours snapshot:

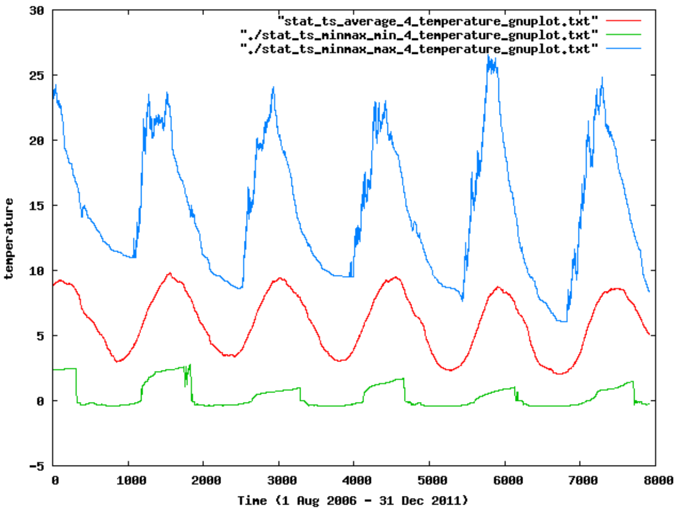


Figure 12: Minimum (green), average (red) and maximum (blue) instantaneous values of temperature in the BS domain, plotted every 6 hour.

2.50 2006.10.28 00:00
2.63 2011.12.27 12:00
2.73 2011.12.26 12:00
2.87 2007.01.10 12:00
2.99 2011.12.27 18:00

i.e. they are associated with three events

2006.10.28
2007.01.10
2011.12.26-27

The two first are consistent with high-water Nos. 300 and 302 since 1702 measured in St. Petersburg[12], reaching 2.24 meter and 2.23 meter, respectively (datum level of the observations can according to our experience be some 25-35 cm lower than the model's datum level in that area, cf. e.g. the modification of the datum levels applied in figures 17 and 18 later in this

[12] www.pices.int/publications/presentations/Climate_Change_2008/Climate_Change_2008_W6/W6_Klevannyy_2.pdf

section). We know from previous storms that the water level sometimes can be somewhat higher in the bottom of the Gulf of Riga as registered at station Pärnu (see e.g. the report on the January 2005 event elsewhere[13]), But during the two events we deal with here, the 2.50 m and 2.87 m occur in the model near St. Petersburg. Unfortunately, DMI's data base of water level observations only has data from St. Petersburg starting on June 2007 and data from the Kronstadt station has a huge gap from 2006.01.27 to 2007.06.13. We must therefore look at other stations in our attempt to verify the events. In figures 13 and 14 we show comparisons between model results and observation at the two Estonian stations Pärnu and Tallinn: From these plots we find the agreement satisfactory and in particular we cannot say that the model over-predicts the water level in the Gulf of Finland, so we are confident that the model results for these two events are not far from the reality.

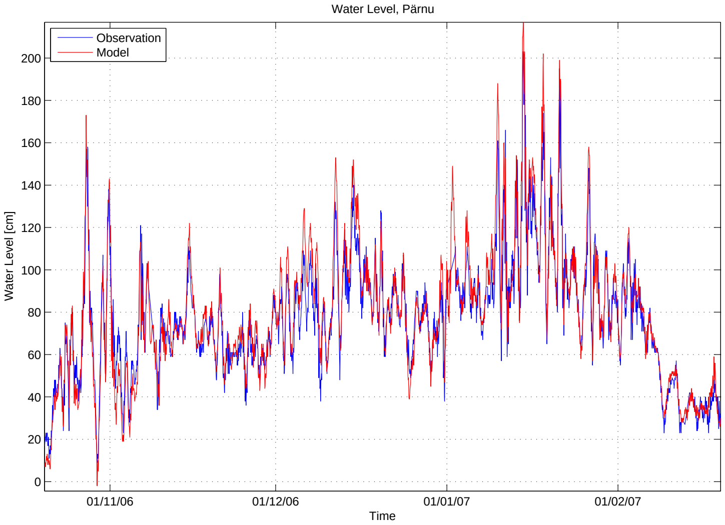


Figure 13: Observed (blue) and modelled (red) water level at Estonian station Pärnu in the bottom of the Gulf of Riga during Oct '06 - Feb '07. An arbitrary modification of +30 cm of the datum level of the observation data has been applied to ease the comparison.

The last and highest of the above-mentioned events is during a known December storm of that year, and we need to confirm the model results with

[13] http://ocean.dmi.dk/case_studies/surges/08jan05.php

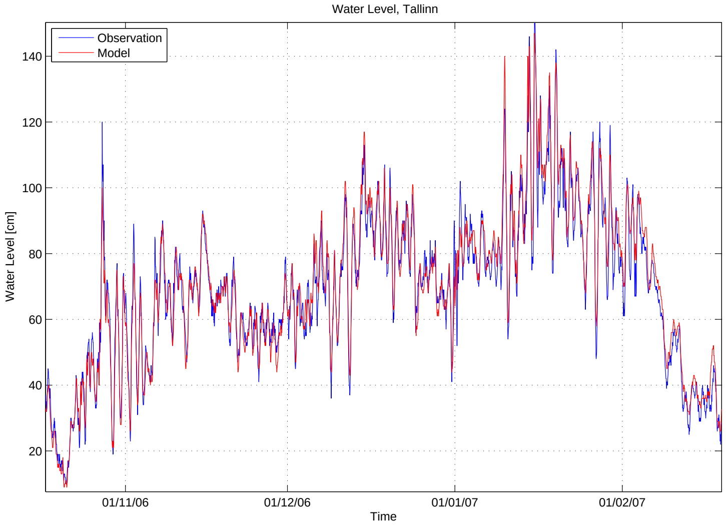


Figure 14: Observed (blue) and modelled (red) water level at Estonian station Tallinn in the Gulf of Finland during Oct '06 - Feb '07. An arbitrary modification of +30 cm the datum level of the observation data has been applied to ease the comparison.

observation data, e.g. from Pärnu or St. Petersburg which are the two station sites usually exposed to high-waters. We do have observations from both these two stations during December 2011, but neither observations nor model results give the above mentioned extremely high water levels at Pärnu, see figure 15 from which we might suspect the model to over-predict the three last peaks. In the model, the 3 m water level occurs in the Neva Bay, but we cannot use the observations at station St. Petersburg to confirm this because of the St. Petersburg Dam which we have not implemented into our model. Actually, the first time *ever* use of this dam to hold back the incoming Baltic water into Neva Bay was during the storm November 28, 2011[14]. We have compared to available observations at the three stations Tallinn, Kronstadt and St. Petersburg in the Gulf of Finland for the whole simulation and found reasonable agreement in general at all three stations (not shown here), also for storm events. If we put focus on the two last months of 2011, we find a reasonable agreement between model results and observations for the November 28[15] and the December 26-27 storms in

[14] http://en.wikipedia.org/wiki/Saint_Petersburg_Dam

[15] for more on the November 27-28 event, see section 5.8

Tallinn, cf. figure 16. In Kronstadt and St. Petersburg, cf. figures 17 and 18, which are both positioned on the inside of the dam we clearly see the effects of the dam for those events. To investigate further, we re-ran the December 2011 using the very latest DMI-HIRLAM model SKA[16] as forcing instead of the S03 model but the results were pretty much similar. The model might possibly over-predict the highest high-water events, but we find the overall performance satisfactory and we find no reason not to continue, even-though we cannot at present verify the model's most extreme high-water levels.

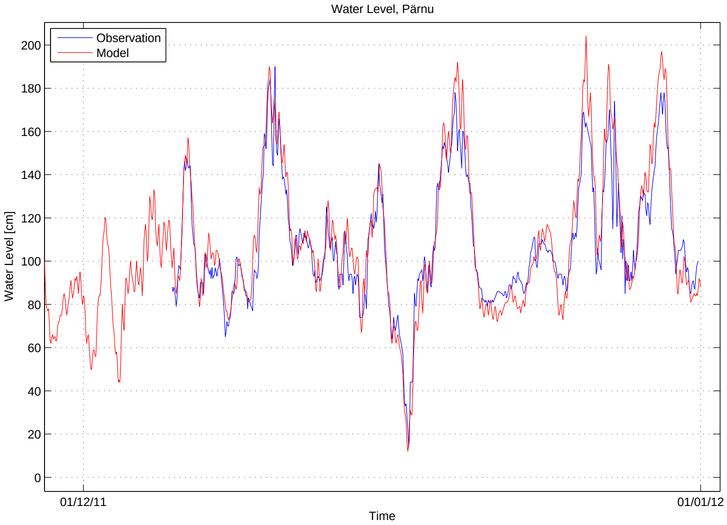


Figure 15: Observed (blue) and modelled (red) water level at Estonian station Pärnu in the bottom of the Gulf of Riga during December 2011. An arbitrary modification of +40 cm of the datum level of the observation data has been applied to ease the comparison.

To some oceanographers the evolution of the modelled average water level in the BS domain (red curve in figure 10) might seem somewhat drastic at a first glance, especially with its relatively large amplitude signal and e.g. the build-up during the 2006 October-November and 2011 November-December storms. It is known in the oceanographic community that the mean water level in the Baltic Sea can be monitored from the measurements at the

[16] The SKA model was put into production during that period and the previous S03 model was being out-phased. Data from SKA was available from 5 December.

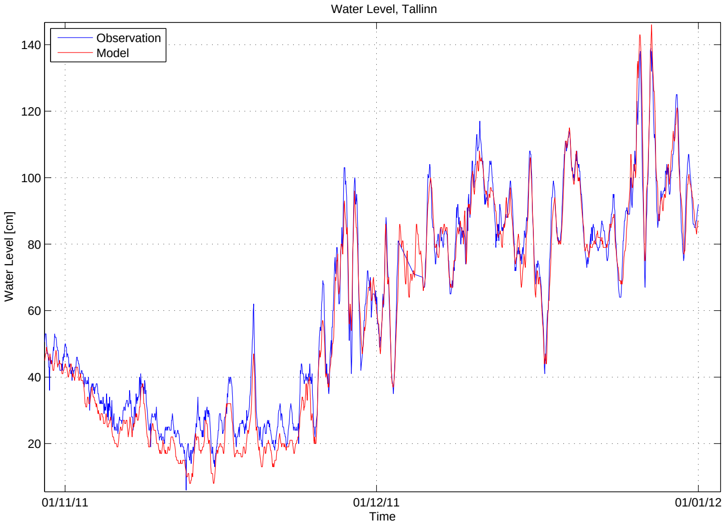


Figure 16: Observed (blue) and modelled (red) water level at Estonian station Tallinn in the Gulf of Finland during November-December 2011. An arbitrary modification of +40 cm the datum level of the observation data has been applied to ease the comparison.

tide gauge station *Landsort* in the central part of the Baltic Sea. A long-term data set is available from PSMSL[17]. In figure 19 we show the monthly mean water level from the Landsort station and compare it to the modelled monthly mean water level for the BS domain (i.e. the monthly averaged values of the red curve in figure 10). It seems quite clear that there is a high degree of correlation between the two data sets. Thus, we feel confident that we do not have artificial trends or unrealistic low-frequent signals introduced into the model.

5.7 Sea levels around Denmark in 2011

The aim of this section is to share initial findings when looking at sealevel results in 2011. Generally, we have tried to make it easy to get a survey of the results by ploting all results on the same map. First, you can see the positions of the Danish stations in figure 49/50 in appendix B and you can

[17] Permanent Service for Mean Sea Level, see
`http://www.psmsl.org/data/obtaining/rlr.monthly.data/2132.rlrdata`

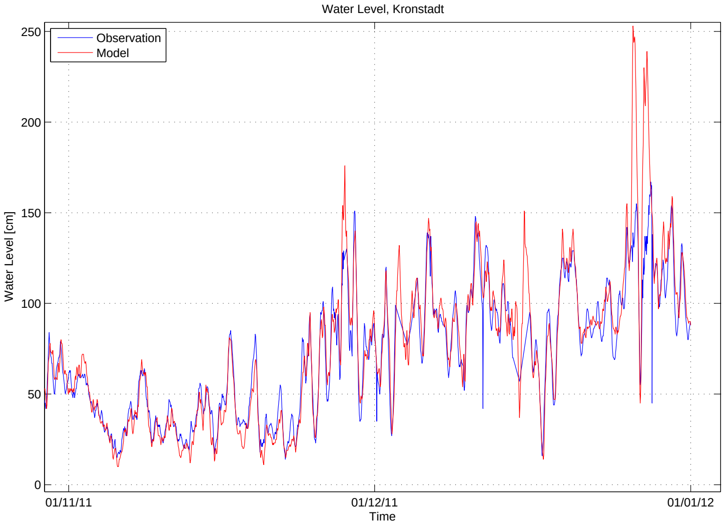


Figure 17: Observed (blue) and modelled (red) water level at Russian station Kronstadt in the Gulf of Finland during November-December 2011. An arbitrary modification of +35 cm the datum level of the observation data has been applied to ease the comparison.

also see the positions where modelled time-series are extracted to be cross-compared with the observations. Note that the positions are not completely identical, e.g. the locations of Skagen, Hanstholm and Tejn seem somewhat inconsistent. Note also that the model setup does not cover all the stations, e.g. stations placed in the Limfjord or in Randers Fjord are not present in this ocean model setup since this model setup does not cover these (and other) fjords.

It should not come as a big surprice that not all observations can be regarded as valid data describing the state of the ocean and thereby being useful as a source for model validation, and we have used a simple screening process[18] to get rid of the worst observations:

- None of the stations will have a sealevel less than -250 cm. The lowest ever measured was -210/-220 cm.
- Maximum deviation on a 10 minute scale is ±40 cm.

[18] Thanks to Jacob Woge Nielsen, DMI, for sharing his process.

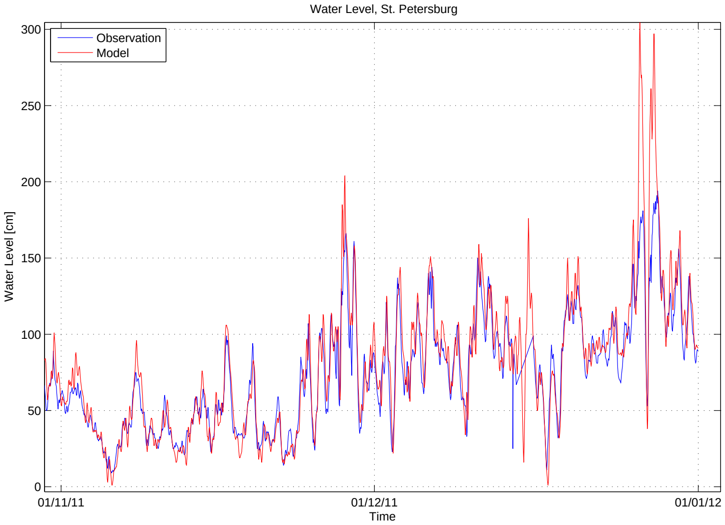


Figure 18: Observed (blue) and modelled (red) water level at Russian station St. Petersburg in the bottom of the Gulf of Finland during November-December 2011. An arbitrary modification of +25 cm of the datum level of the observation data has been applied to ease the comparison.

- Maximum sea level

  500 cm: 26359, 26361, 26346, 26136, 26137, 26143, 26144, 25236, 25347, 25343, 25344, 25147, 25149

  400 cm: 24342, 24343, 24122, 24123, 24018, 24006

  250 cm: the rest of the stations

  200 cm: 32048, 32096

- No similiar measurements three times in a row, i.e. measurements must change in order to be considered valid in the data set of observations

Using this screening on the observations we obtain an observation coverage as shown in figure 51 in appendix B. Figure 54 shows the sealevel range measured throughout 2011 for the screened sealevels and figure 55 shows the sealevel range that the model produced . These figures cleary show that the screening procedure above is insufficient, e.g. Hals Barre shows -249 cm.

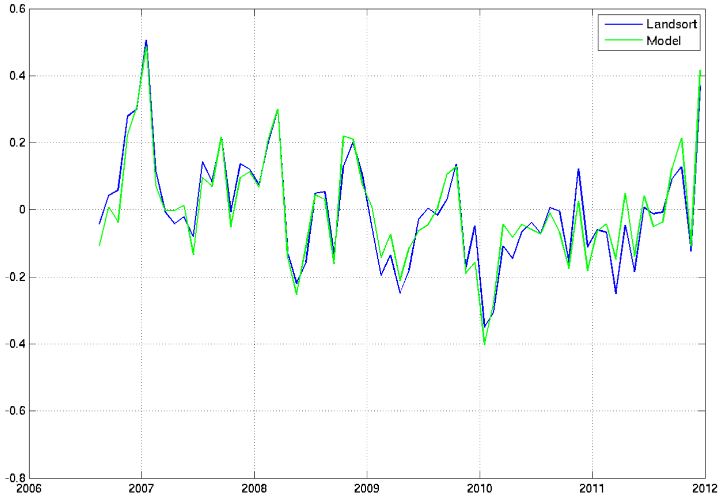


Figure 19: Monthly mean water levels. Blue curve is from observations at station Landsort. Green curve is the monthly mean value of the spatially averaged BS z from the model results. For better visual comparison both curves have been processed to having zero mean value over the shown 65 months long period.

When we cross-compare the screened observations with the model results using simple statistics like:

$$bias = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i) \tag{1}$$

$$std = \sqrt{\frac{1}{(N-1)}\sum_{i=1}^{N}(x_i - y_i - bias)^2} \tag{2}$$

with x_i being model outcome and y_i being the screened observation at timepoint i where we apply a simple and straight-forward matching procedure[19] and we obtain the results in figure 52 in appendix B. Moreover, figure 53

[19] The fact that the timestamps and positions in the observations do not map exactly one-to-one to our model outcome requires a strategy for time and space interpolation. We have chosen to take the most simple approach, i.e. for timestamp interpolation - if there exists a screened observation within an interval of ± 5 minutes around the timepoint i

in appendix B shows the hit-rate on each station when we define a hit to be a residual between a screened observation and the model to be less than 20 cm. The 20 cm coincides with the uncertainty on the measurement itself.

5.8 Storm surge issue in Øresund

The storm during the 27th and 28th November 2011 gave rise to storm surges in the IDW. At several stations the water level reached about a 50-year event[20], i.e. in Øresund with nearly 1.4 meter at the Copenhagen tide gauge station KBH. There were two events: During the first event the water was pulled away eastward from the southern part of IDW and south-western Baltic while water was pushed in from north. This gave huge gradients over short distances at the narrow passages in IDW, e.g. up to 2.5 meter difference between the KBH and Drogden stations. The second event was high water everywhere in IDW. This came as a valuable opportunity test for our models, especially the large gradients during the first event should put some severe demands on the model's capability. We often have storm surge in the North Sea - Baltic Sea region, but it is not very often that we have extreme high-water events in the Danish Straits[21].

Our models predicted the two events very well everywhere (not shown here) except in the northern part of Øresund where the KBH station is located. At the KBH station the first peak was almost completely absent in the model described here as well as in the forecast models DMI ran in production at that time (i.e. the storm surge model and the MyOcean V2) which were all based on the same bathymetry and grid in the area in and around Øresund.

It is well-known that the 1 n.m. resolution can be too coarse to resolve the narrow Danish straits and the flow therein, unless some kind of parametrization is performed. Since the model produces nice water levels just north of Øresund in the northern entrance to Øresund and south of the Drogden Sill, we can suspect either of the following cross-section and resistance issues or combinations hereof: i) the northern entrance does not allow enough water

of the model then we use that one; and for space interpolation we have not done any interpolation but assumes that the placements are consistent which we have seen is not the case for all stations.

[20] http://www.dmi.dk/dmi/vinden_valtede_vandet

[21] the last similar, though not quite as severe, case was on 1st November 2006, cf. http://ocean.dmi.dk/case_studies/surges/01nov06.php

sufficiently quickly into Øresund, ii) the Drogden Sill area does not represent sufficient resistance against southward outflow from Øresund to hold back the water, iii) the bottom friction in the area is too low.

We have tested these issues on the 27-28 November, 2011, storm surges and found, cf. figure 20, in order of significance, that our model was indeed too shallow and narrow around the northern entrance with too large resistance against incoming water, that the bottom friction could be increased locally in northern Øresund to give higher amplitude on the first peak at KBH and obtain larger gradient across the Drogden Sill, and that the water level prediction could benefit from a slightly more narrow cross section in the Drogden Channel, *without destroying the good signal on other stations.* As the figure shows small local adjustments can improve the modelled water level significantly.

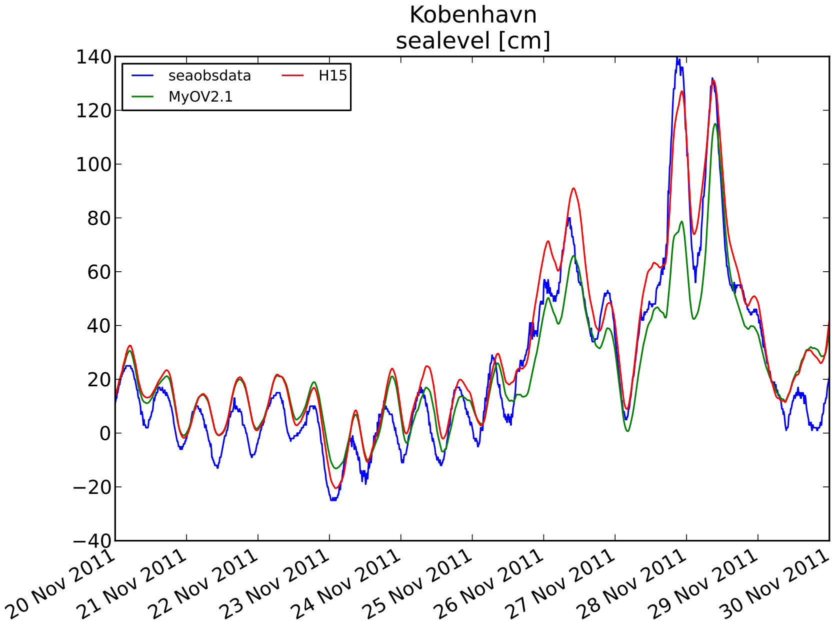


Figure 20: The November 2011 storm surge events as seen from station KBH. Observation from the data base (blue), original model result from the MyOcean V2.1 run (green), and model result from a scenario (H15) after some experimentation with adjusting the bathymetry and bottom friction locally (red).

6 Variant0

This testcase is a beta version of the upcoming MyOcean setup (likely to be named MyO V3). The resolution in the Baltic Sea is now 1 n.m. and we have also improved both initial field and the bathymetry as well as increased the vertical resolution in the deeper parts for this subdomain. Table 21 summarizes the setup and figure 2 shows how the four sub-domains nest to each other. The I_r number for this setup is 60.6. This section will present the findings that we have done with this setup. We have done some simple scaling studies on our local cray XT5 installation as well as on the cray XE6 installation at CSCS. Moreover, we tried to present the time step view of the overall algorithm and and zoomed in on the major IO activities to get reference numbers in places prior to the planned IO rewrite in the future. Finally, we have tried to measure the attained performance with different number of passive tracers.

	IDW	BS	WS	NS
resolution [n.m.]	0.5	1.0	1.0	3.0
mmx [N/S]	482	720	149	348
nmx [W/E]	396	567	156	194
kmx	77	122	24	50
gridpoints	14697144	49805280	557856	3375600
iw2	80884	119334	11581	18908
iw3	1583786	6113599	103441	479081
f_{iw3}	10.8%	12.3%	18.5%	14.2%
φ [latitude]	57 35 45N	65 53 30N	55 41 30N	65 52 30N
λ [longitude]	09 20 25E	14 35 50E	06 10 50E	04 07 30W
$\Delta\varphi$	0 0 30	0 1 00	0 1 00	0 3 00
$\Delta\lambda$	0 0 50	0 1 40	0 1 40	0 5 00
dt [sec]	12.5	12.5	25	25
maxdepth [m]	78.00	398.00	53.60	696.25
min Δx	827.62	1261.65	1740.97	3787.40
CFL	0.790	0.910	0.626	0.797
I_r	12.0	46.3	0.4	1.8

Table 21: The testcase termed variant0. The I_r number for the setup is 60.6.

6.1 Short simulations

Figure 21 and figure 22 shows the scaling on our local cray XT5 and the cray XE6 at CSCS, respectively.

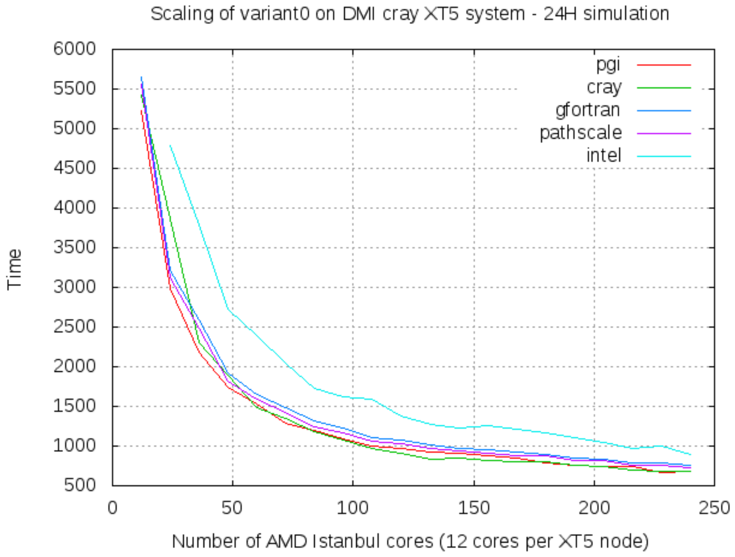


Figure 21: Scaling of variant0 without passive tracers using automatically generated I-slices performed on our local Cray XT5 with 12 openMP threads on each MPI task and one MPI task on each node. Note that the 20 timings for each compiler above gave rise to identical md5sums for the `restart` file.

Here we present a short summary of the profiles obtained with the PGI compiler. Figure 23 shows a snapshot of a serial profile whereas figure 24 shows a snapshot of a threaded profile (1 MPI task, 12 threads) and finally figure 25 shows a snapshot of a threaded MPI profile (20 tasks, 12 threads).

6.2 Passive tracer performance

We have tried to evaluate how much it would cost to run with an additional number of passive tracers. For this purpose, we have added a configure option `--enable-bench-coupler` that allows one to run `tflow` with any number of passive tracers. We have confined ourselves to run with 9 (fig-

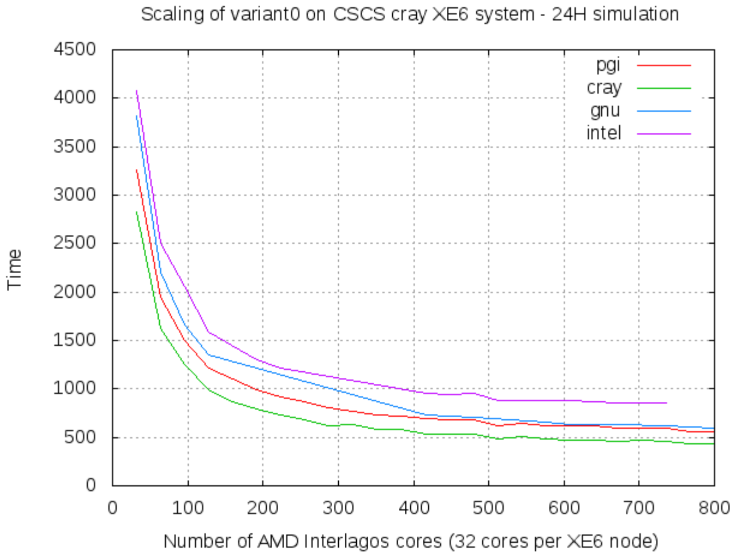


Figure 22: Scaling of variant0 without passive tracers using automatically generated I-slices performed on the Cray XE6 system at CSCS with 32 openMP threads on each MPI task and one MPI task on each node. Note that the 20 timings for each compiler above gave rise to identical md5sums for the `restart` file. Even for the PGI compiler. The PGI compiler version used on CSCS is the same as the one used locally, i.e. 11.10.0.

ure 26 and figure 27) and 25 passive tracers (figure 28). We found that one ran out of memory on our local XT5 system when running with many tasks (say above 35) and 9 tracers and that one would run out of memory when running with more than 12 tasks and 25 tracers. Thus, the scaling study of the runs with 25 tracers have been done only on the XE6 system.

It is worth mentioning that on the CNL systems the kernels are configured to allow over-commit of memory so for those compilers that have not written their own memory allocator the applications will die with `OOM` on CNL, c.f. the kernel pseudo-file:

```
/proc/sys/vm/overcommit_memory
```

	pgi	cray	pathscale	gnu	intel
Initialisation	28.9/18.7	27.5	25.0	38.8	26.8/15.9
IO bathymetri	/2.8	13.3/1.0	8.5/1.6	22.0/4.0	11.5/1.1
IO tempdat	14.0		8.3	18.0	12.5
IO restart	3.9		3.4	5.0	3.9

Table 22: Timings of the initialisation and from the most expensive IO parts using different compilers. The two numbers in the initialisation/bathymetri line are for ascii vs. binary fortran files for storing the bathymetri.

```
|| Time%  |        Time  |  Calls  |Group
||  34.9% | 3344.060794 | 5184.0 |momeqs_wrapping_momeqs_default_
||  34.4% | 3289.960885 | 2304.0 |tflow_tflow_int_
||  13.5% | 1289.936352 | 1152.0 |turbmodels_turbmodel_
||   3.8% |  361.790890 | 5184.0 |smagorinsky_smag_
||   3.0% |  285.949189 |  864.0 |cmod_hydrodynamics_solvehydrodynamics_
||   2.4% |  226.756856 | 5184.0 |smagorinsky_deform_
||   2.2% |  213.894891 | 5184.0 |masseqs_masseqs_solver_z_
||   1.5% |  140.641739 | 1156.0 |thermodynamic_thermodyn_1_
||   1.4% |  133.531607 | 1156.0 |cmod_dens_dens_
```

Figure 23: Serial profile of a 6H simulation for Variant0, PGI, tune flag

```
|| Time%  |        Time  | Calls  |Group
||-------------------------------------------------------------------------------------------------
||  25.0% |  340.665730 | 5184.0 |momeqs_wrapping_momeqs_default_
||   9.8% |  133.348520 | 1152.0 |tflow_tflow_int_.REGION@li.2433
||   8.8% |  119.664981 | 1152.0 |turbmodels_turbmodel_
||   6.5% |   88.235754 | 5184.0 |cmod_hydrodynamics_solvemasseq_.REGION@li.564
||   6.4% |   86.570924 | 1152.0 |tflow_tflow_int_.REGION@li.2473
||   5.0% |   68.631243 | 1152.0 |tflow_tflow_int_.REGION@li.2487
||   4.1% |   55.351749 | 5184.0 |smagorinsky_deform_
||   3.6% |   48.823833 | 5184.0 |smagorinsky_smag_
||   3.5% |   47.114687 | 1152.0 |tflow_tflow_int_.REGION@li.2500
||   3.3% |   45.017830 | 1152.0 |tflow_tflow_int_.REGION@li.2604
||   3.0% |   41.486708 | 1152.0 |tflow_tflow_int_.REGION@li.2518
||   2.8% |   38.635646 | 1152.0 |tflow_tflow_int_.REGION@li.2456
||   2.8% |   38.423146 | 5184.0 |masseqs_masseqs_solver_z_
||   1.3% |   18.389167 | 3456.0 |dmi_mpi_dmpi_gather_copy_cmp_
||   1.3% |   17.722863 |  577.0 |MAIN_
||   1.1% |   14.697439 | 1156.0 |thermodynamic_thermodyn_1_
||   1.1% |   14.591464 | 5184.0 |dmi_mpi_dmpi_gather_copy_
||=================================================================================================
|   3.0% |   40.626639 |       -- |     -- |  51307.0 |OMP
|   0.0% |    0.000800 | 0.000000 |   0.0% |     11.0 |PTHREAD
|   0.0% |    0.000008 |       -- |     -- |      2.0 |MPI
|==================================================================================================
```

Figure 24: openmp_mpi, 12 threads, 1 task profile of a 6H simulation for Variant0, PGI, tune flag

```
|| Time%  |       Time |   Calls  |Group
|| Time%  |       Time |     Imb. |  Imb.   |    Calls   |Group
||        |            |     Time | Time%   |            | Function
|| 100.0% | 200.430665 |       -- |      -- | 592591.6 |Total
|----------------------------------------------------------------------------------------------------
|!  95.9% | 192.285315 |       -- |      -- | 537578.9 |USER
||---------------------------------------------------------------------------------------------------
||  20.0% |  40.106402 | 6.889583 |  15.4% |  59332.0 |dmi_mpi_dmpi_halo_nonblocking_
||  16.2% |  32.435089 | 6.330884 |  17.2% |  20744.0 |dmi_mpi_dmpi_distribute_halo_nc_
||   7.5% |  15.074943 | 1.293811 |   8.3% |   5184.0 |momeqs_wrapping_momeqs_default_
||   5.0% |  10.065974 | 2.838470 |  23.2% |   3866.0 |dmi_mpi_dmpi_gather_all_
||   4.7% |   9.365783 | 3.231184 |  27.0% |   3456.0 |dmi_mpi_dmpi_gather_copy_cmp_
||   4.5% |   9.031637 | 0.757797 |   8.1% |      6.0 |dmi_mpi_dmpi_broadcast_met_info_
||   3.1% |   6.268156 | 1.056853 |  15.2% |   1152.0 |turbmodels_turbmodel_
||   3.0% |   6.066532 | 0.190502 |   3.2% |      1.0 |restart_readrestart_
||   2.9% |   5.763540 | 0.135461 |   2.4% |   1152.0 |tflow_tflow_int_.REGION@li.2433
||   2.2% |   4.441544 | 2.720642 |  40.0% |   5184.0 |dmi_mpi_dmpi_gather_copy_
||   2.2% |   4.375008 | 0.578508 |  12.3% |      4.0 |cmod_params_getparams_
||   1.7% |   3.459045 | 0.173162 |   5.0% |   1152.0 |tflow_tflow_int_.REGION@li.2473
||   1.5% |   3.013457 | 0.535040 |  15.9% |   1152.0 |tflow_tflow_int_.REGION@li.2487
||   1.4% |   2.879061 | 0.090271 |   3.2% |      1.0 |exit
||   1.4% |   2.821230 | 0.143194 |   5.1% |   5184.0 |cmod_hydrodynamics_solvemasseq_.REGION@li.564
||   1.3% |   2.607475 | 4.288471 |  65.5% |   8640.0 |dmi_mpi_dmpi_gather_mcf_nb_
||   1.2% |   2.427550 | 0.294507 |  11.4% |   5184.0 |smagorinsky_deform_
||   1.2% |   2.364569 | 2.142709 |  50.0% |    864.0 |dmi_mpi_dmpi_barrier_
||   1.0% |   2.017156 | 0.119113 |   5.9% |   1152.0 |tflow_tflow_int_.REGION@li.2604
||   1.0% |   2.013952 | 0.335887 |  15.0% |   5184.0 |masseqs_masseqs_solver_z_
||===================================================================================================
|   2.4% |   4.803867 |       -- |      -- |  54999.6 |OMP
||---------------------------------------------------------------------------------------------------
|   1.4% |   2.876969 | 0.276006 |   9.2% |      1.0 | dmi_omp_domp_init_.REGION@li.46(ovhd)
||===================================================================================================
|   1.7% |   3.340686 |       -- |      -- |      2.0 |MPI
||---------------------------------------------------------------------------------------------------
|   1.7% |   3.340684 | 0.202856 |   6.0% |      1.0 | mpi_finalize
||===================================================================================================
```

Figure 25: openmp_mpi, 12 threads, 20 task profile of a 6H simulation for Variant0, PGI, tune flag

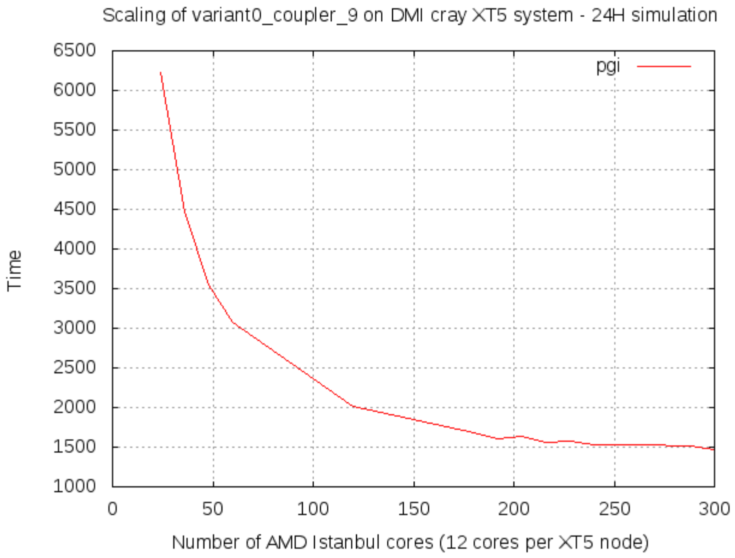


Figure 26: Scaling of variant0 with 9 passive tracers using automatically generated I-slices performed on our local Cray XT5 with 12 openMP threads on each MPI task and one MPI task on each node.

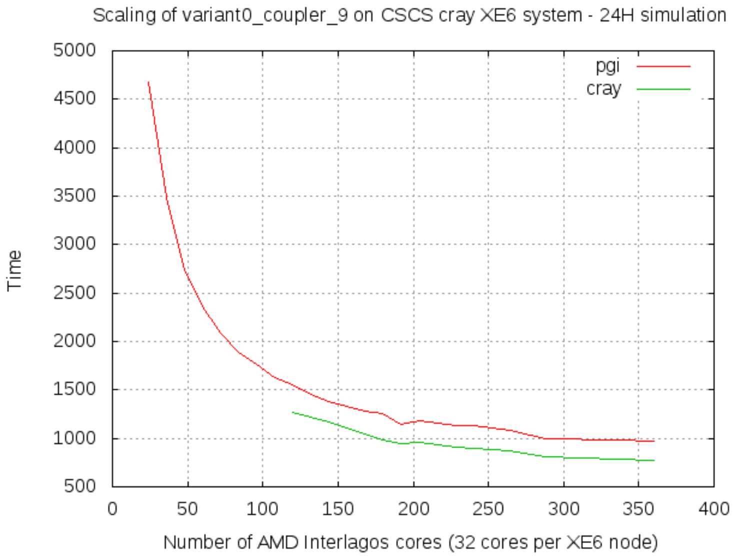


Figure 27: Scaling of variant0 with 9 passive tracers using automatically generated I-slices performed on the Cray XE6 with 32 openMP threads on each MPI task and one MPI task on each node.

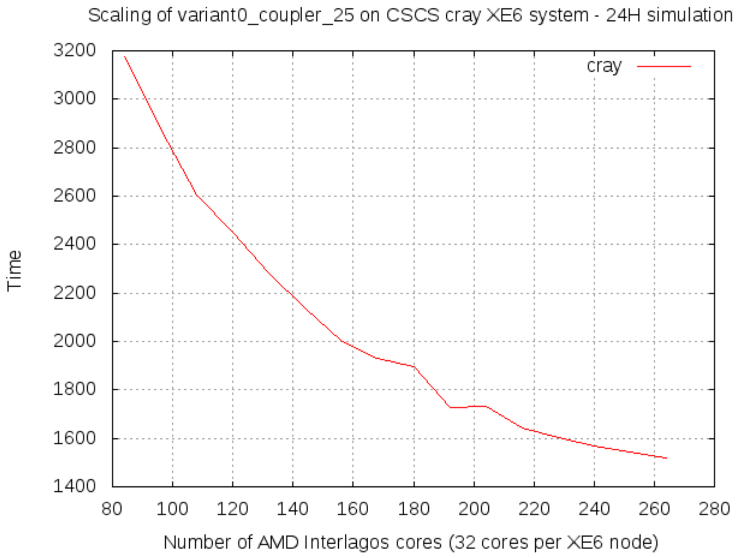


Figure 28: Scaling of variant0 with 25 passive tracers using automatically generated I-slices performed on the Cray XE6 with 32 openMP threads on each MPI task and one MPI task on each node.

7 Variant1

Table 23 summarizes the setup and figure 29 shows how the four sub-domains nest to each other. The I_r number for this setup is 89.6 and the scaling attained (as is) is shown in figure 30.

	IDW	**BS**	**WSNS**	**rNS**
resolution [n.m.]	0.5	1.0	1.0	3.0
mmx [N/S]	482	720	389	220
nmx [W/E]	396	567	208	127
kmx	77	122	110	46
gridpoints	14697144	49805280	8772716	1285240
iw2	80884	119334	32515	15593
iw3	1583786	6113599	1560598	409049
f_{iw3}	10.8%	12.3%	17.8%	31.8%
φ [latitude]	57 35 45N	65 53 30N	59 41 30N	59 28 30N
λ [longitude]	09 20 25E	14 35 50E	06 10 50E	04 07 30W
$\Delta\varphi$	0 0 30	0 1 00	0 1 00	0 3 00
$\Delta\lambda$	0 0 50	0 1 40	0 1 40	0 5 00
dt [sec]	10	10	10	20
maxdepth [m]	78.00	398.00	696.25	336.96
min Δx	827.62	1261.65	1558.76	4706.49
CFL	0.632	0.728	0.957	0.441
I_r	15.0	57.9	14.8	1.9

Table 23: The testcase termed variant1. The I_r number for the setup is 89.6.

We show a few snapshots of profiles below. All the profiles shown below are done using PGI generated binaries.

7.1 Performance profiles

Here we present a short summary of the profiles obtained with the PGI compiler. Figure 33 shows a snapshot of a serial profile whereas figure 34 shows a snapshot of a threaded profile (1 MPI task, 12 threads) and finally figure 35 shows a snapshot of a threaded MPI profile (20 tasks, 12 threads).

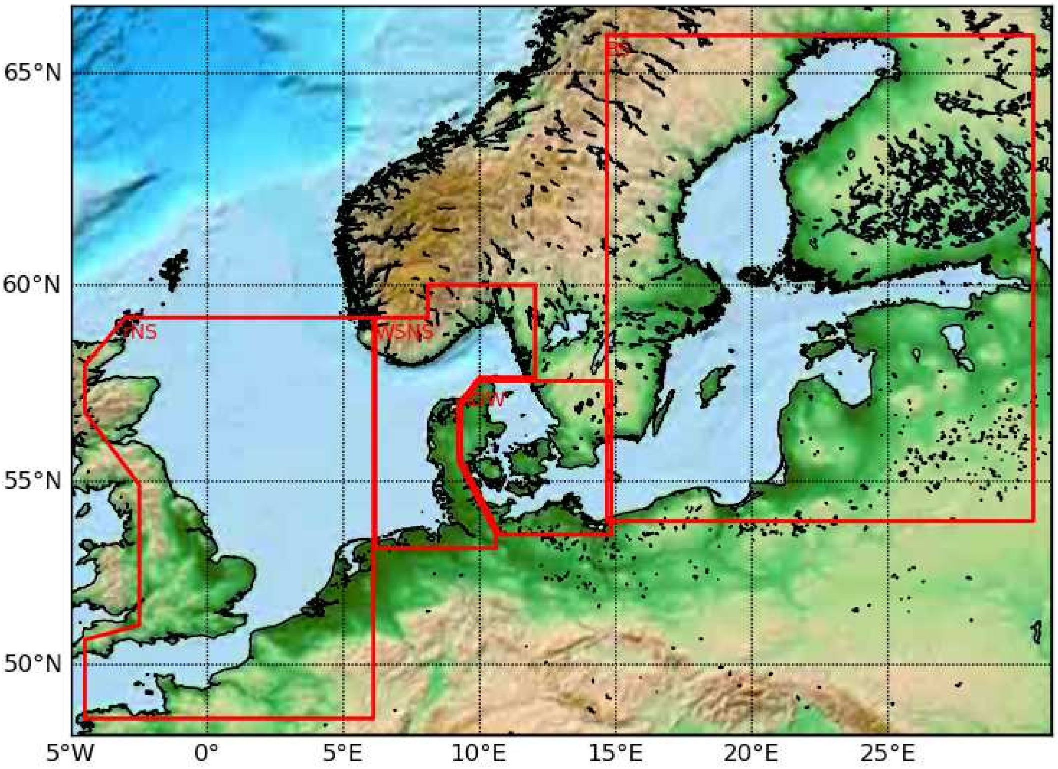


Figure 29: Nesting of the four domains in the variant1 case and the variant2 case.

	pgi	cray	pathscale	gnu	intel
Initialisation	/20.9	31.9/	31.1/	45.1/	33.0/18.5
IO bathy	/3.0	15.2/1.1	10.5/1.8	24.4/4.2	13.2/1.5
IO tmpdat	16.0	12.1	10.8	21.1	12.4
IO restart	4.3	3.1	3.6	5.7	3.2

Table 24: Timings of the serial IO activities and also the initialization prior to the timeloop emerging from different compilers. Note that *IO bathymetry* is part of the *Initialisation* time. The first number in the first two lines is for ascii bathymetry files whereas the second timings is for binary bathymetry files. Note that we have not tried to run all combinations.

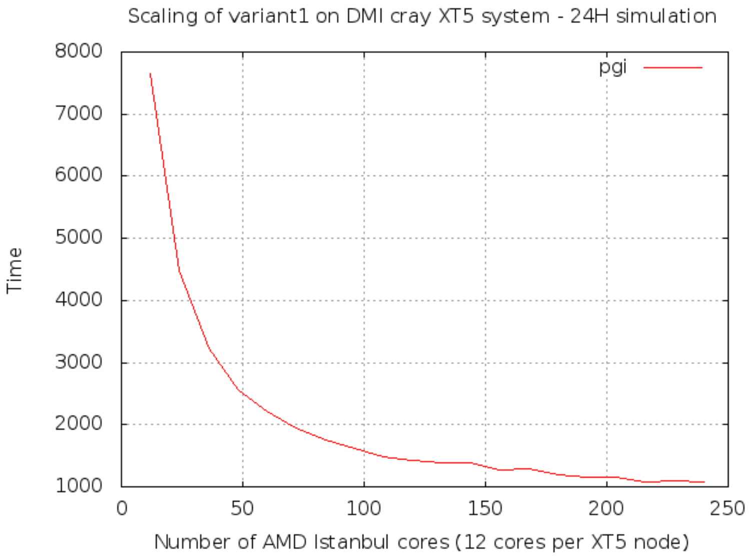


Figure 30: Scaling of the variant 1 using automatically generated I-slices performed on our local Cray XT5 with 12 openMP threads on each MPI task and one MPI task on each node.

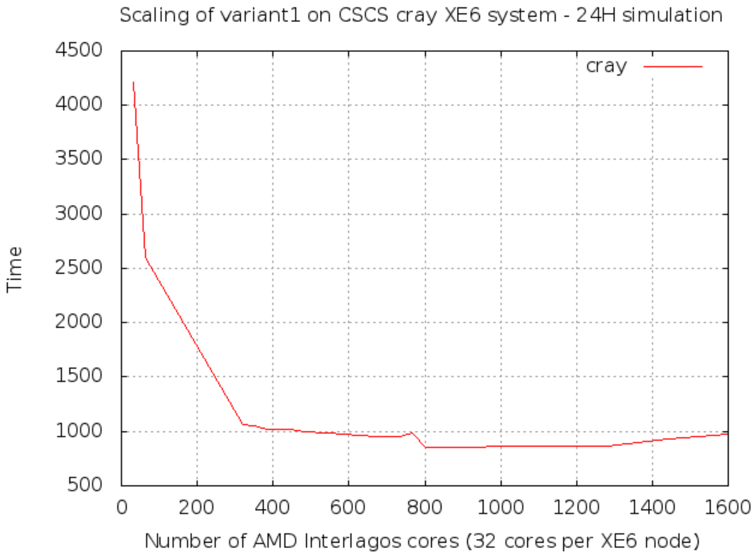


Figure 31: Scaling of variant1 without passive tracers using automatically generated I-slices performed on the Cray XE6 system at CSCS with 32 openMP threads on each MPI task and one MPI task on each node. Note that the 20 timings for each compiler above gave rise to identical md5sums for the `restart` file. Even for the PGI compiler. The PGI compiler version used on CSCS is the same as the one used locally, i.e. 11.10.0.

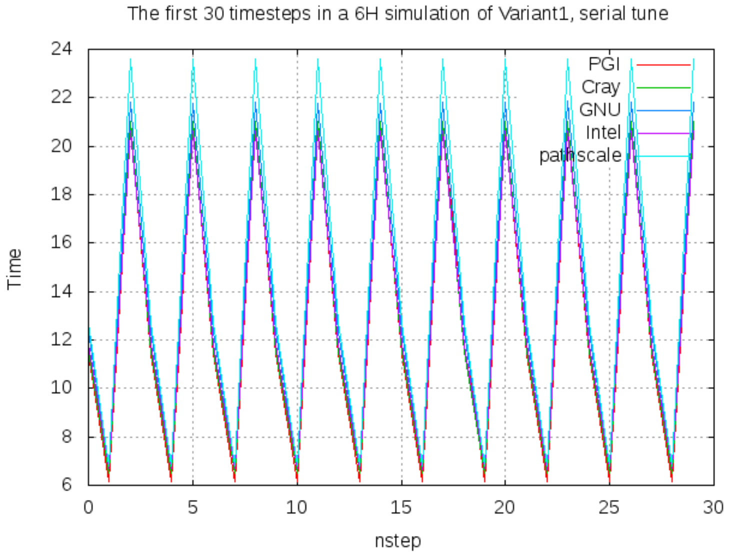


Figure 32: Serial timing of the individual time steps in the Variant1 testcase.

```
||   35.0% |   4832.201451 | 0.000000 |    0.0% |      7560.0 |momeqs_wrapping_momeqs_default_
||   34.2% |   4715.862602 | 0.000000 |    0.0% |      2880.0 |tflow_tflow_int_
||   13.5% |   1862.331507 | 0.000000 |    0.0% |      1440.0 |turbmodels_turbmodel_
||    3.8% |    530.070468 | 0.000000 |    0.0% |      7560.0 |smagorinsky_smag_
||    3.1% |    430.994808 | 0.000000 |    0.0% |      1080.0 |cmod_hydrodynamics_solvehydrodynamics_
||    2.4% |    334.680119 | 0.000000 |    0.0% |      7560.0 |smagorinsky_deform_
||    2.3% |    313.201253 | 0.000000 |    0.0% |      7560.0 |masseqs_masseqs_solver_z_
||    1.4% |    198.135918 | 0.000000 |    0.0% |      1444.0 |thermodynamic_thermodyn_1_
||    1.4% |    194.307058 | 0.000000 |    0.0% |      1444.0 |cmod_dens_dens_
```

Figure 33: Serial profile of a 6H simulation for Variant1, PGI, tune flag

```
|| Time%  |        Time  | Calls   |Group
||-----------------------------------------------------------------------------------------------
||  25.2% |  494.773661 | 0.000000 |    0.0% |   7560.0 |momeqs_wrapping_momeqs_default_
||   9.8% |  192.398774 | 0.000000 |    0.0% |   1440.0 |tflow_tflow_int_.REGION@li.2433
||   8.8% |  172.813665 | 0.000000 |    0.0% |   1440.0 |turbmodels_turbmodel_
||   6.6% |  130.562263 | 0.000000 |    0.0% |   7560.0 |cmod_hydrodynamics_solvemasseq_.REGION@li.564
||   6.4% |  125.803213 | 0.000000 |    0.0% |   1440.0 |tflow_tflow_int_.REGION@li.2473
||   5.2% |  101.932754 | 0.000000 |    0.0% |   1440.0 |tflow_tflow_int_.REGION@li.2487
||   4.1% |   80.877956 | 0.000000 |    0.0% |   7560.0 |smagorinsky_deform_
||   3.6% |   70.578510 | 0.000000 |    0.0% |   7560.0 |smagorinsky_smag_
||   3.6% |   70.011629 | 0.000000 |    0.0% |   1440.0 |tflow_tflow_int_.REGION@li.2500
||   3.4% |   66.143257 | 0.000000 |    0.0% |   1440.0 |tflow_tflow_int_.REGION@li.2604
||   3.2% |   62.224711 | 0.000000 |    0.0% |   1440.0 |tflow_tflow_int_.REGION@li.2518
||   2.9% |   56.965955 | 0.000000 |    0.0% |   1440.0 |tflow_tflow_int_.REGION@li.2456
||   2.9% |   56.503301 | 0.000000 |    0.0% |   7560.0 |masseqs_masseqs_solver_z_
||   1.3% |   25.418023 | 0.000000 |    0.0% |   7560.0 |dmi_mpi_dmpi_gather_copy_
||   1.1% |   21.796339 | 0.000000 |    0.0% |   4320.0 |dmi_mpi_dmpi_gather_copy_cmp_
||   1.0% |   19.895360 | 0.000000 |    0.0% |   1444.0 |thermodynamic_thermodyn_1_
||   1.0% |   18.788083 | 0.000000 |    0.0% |   1444.0 |cmod_dens_dens_
||===============================================================================================
|   2.8% |   54.700627 |        -- |      -- |  71419.0 |OMP
|   0.0% |    0.000764 | 0.000000 |    0.0% |     11.0 |PTHREAD
|   0.0% |    0.000008 |        -- |      -- |      2.0 |MPI
|================================================================================================
```

Figure 34: openmp_mpi, 12 threads, 1 task profile of a 6H simulation for Variant1, PGI, tune flag

```
|| Time%  |       Time |   Calls   |Group
|| Time%  |       Time |     Imb. |  Imb.   |   Calls    |Group
||        |            |     Time | Time%   |            | Function
|| 100.0% | 317.862961 |       -- |      -- | 831621.3   |Total
|------------------------------------------------------------------------------------------------
|!  97.0% | 308.217329 |       -- |      -- | 755789.9   |USER
||-----------------------------------------------------------------------------------------------
||  25.0% |  79.324438 | 13.135650 |  15.0% |  87124.0 |dmi_mpi_dmpi_halo_nonblocking_
||  17.9% |  57.010310 |  6.714954 |  11.1% |  25928.0 |dmi_mpi_dmpi_distribute_halo_nc_
||   6.9% |  22.009895 |  2.216656 |   9.6% |   7560.0 |momeqs_wrapping_momeqs_default_
||   4.8% |  15.104692 |  4.376259 |  23.6% |   4320.0 |dmi_mpi_dmpi_gather_copy_cmp_
||   3.7% |  11.879094 |  0.125675 |   1.1% |      1.0 |restart_readrestart_
||   3.4% |  10.649024 |  0.818472 |   7.5% |      6.0 |dmi_mpi_dmpi_broadcast_met_info_
||   3.3% |  10.390019 | 14.127070 |  60.7% |   7560.0 |dmi_mpi_dmpi_gather_copy_
||   3.2% |  10.281579 |  2.246290 |  18.9% |   4610.0 |dmi_mpi_dmpi_gather_all_
||   2.8% |   8.999333 |  1.116842 |  11.6% |   1440.0 |turbmodels_turbmodel_
||   2.6% |   8.240736 |  0.236056 |   2.9% |   1440.0 |tflow_tflow_int_.REGION@li.2433
||   1.6% |   5.244724 |  0.477431 |   8.8% |      4.0 |cmod_params_getparams_
||   1.6% |   5.116321 | 10.766107 |  71.4% |  12960.0 |dmi_mpi_dmpi_gather_mcf_nb_
||   1.5% |   4.738231 |  0.280837 |   5.9% |   1440.0 |tflow_tflow_int_.REGION@li.2473
||   1.3% |   4.161589 |  0.130790 |   3.2% |   1440.0 |tflow_tflow_int_.REGION@li.2487
||   1.3% |   4.011499 |  0.150553 |   3.8% |   7560.0 |cmod_hydrodynamics_solvemasseq_.REGION@li.564
||   1.1% |   3.583502 |  0.526550 |  13.5% |   7560.0 |smagorinsky_deform_
||   1.0% |   3.262494 |  4.298877 |  59.8% |   1080.0 |dmi_mpi_dmpi_barrier_
||   1.0% |   3.173445 |  0.489119 |  14.1% |   7560.0 |masseqs_masseqs_solver_z_
||===============================================================================================
|   1.8% |   5.876364 |        -- |      -- |  75818.4 |OMP
|   1.2% |   3.768481 |        -- |      -- |      2.0 |MPI
||-----------------------------------------------------------------------------------------------
|   1.2% |   3.768479 |  0.217877 |   5.8% |      1.0 | mpi_finalize
||===============================================================================================
|   0.0% |   0.000788 |  0.000136 |  15.5% |     11.0 |PTHREAD
|================================================================================================
```

Figure 35: openmp_mpi, 12 threads, 20 task profile of a 6H simulation for Variant1, PGI, tune flag

8 Variant2

Table 25 summarizes the setup and figure 29 shows how the four sub-domains nest to each other. The I_r number for this setup is 227.7 and the scaling attained (as is) is shown in figure 36.

	IDW	**BS**	**WSNS**	**rNS**
resolution [n.m.]	0.25	1.0	1.0	1.0
mmx [N/S]	964	720	389	660
nmx [W/E]	792	567	208	381
kmx	77	122	110	46
gridpoints	58788576	49805280	8900320	11567160
iw2	323536	119334	32515	15593
iw3	6335114	6113599	1560598	409049
f_{iw3}	10.78 %	12.4%	17.8	31.8%
φ [latitude]	57 35 52.5N	65 53 30N	59 41 30N	59 29 30N
λ [longitude]	09 20 12.5E	14 35 50E	06 10 50E	04 09 10W
$\Delta\varphi$	0 0 15	0 1 00	0 1 00	0 1 00
$\Delta\lambda$	0 0 25	0 1 40	0 1 40	0 1 40
dt [sec]	5	10	10	10
maxdepth [m]	78.0	398.0	696.25	336.96
min Δx	413.79	1261.65	1558.76	1568.05
CFL	0.632	0.728	0.957	0.662
I_r	120.1	57.9	14.8	34.9

Table 25: The testcase termed variant2. The I_r number for the setup is 227.7.

8.1 Performance profiles

Here we present a short summary of the profiles obtained with the PGI compiler. Figure 39 shows a snapshot of a threaded profile (1 MPI task, 12 threads) and figure 40 shows a snapshot of a threaded MPI profile (20 tasks, 12 threads).

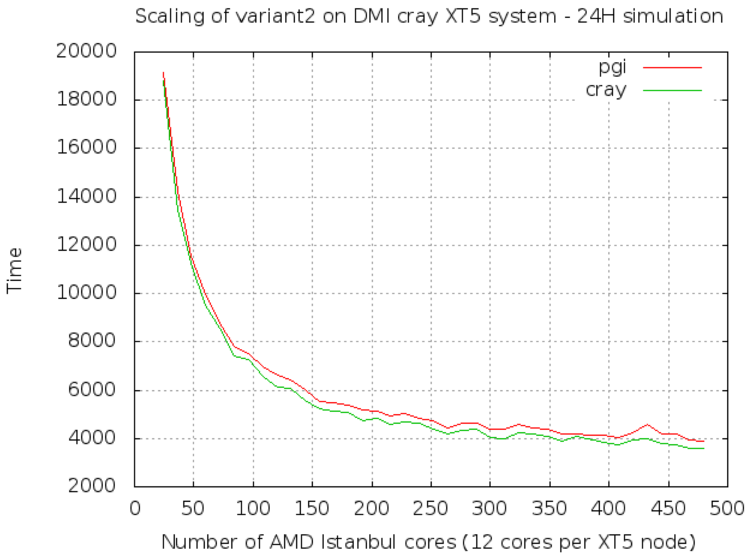


Figure 36: Scaling of the variant2 using automatically generated I-slices performed on our local Cray XT5 with 12 openMP threads on each MPI task and one MPI task on each node.

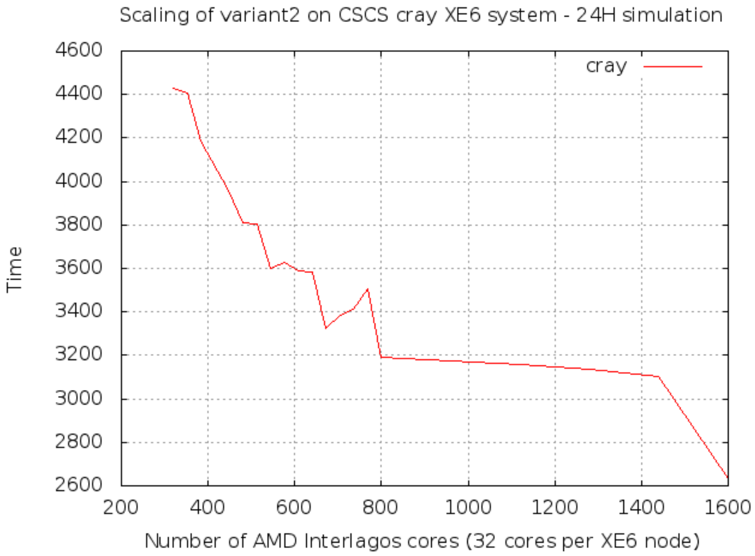


Figure 37: Scaling of variant2 without passive tracers using automatically generated I-slices performed on the Cray XE6 system at CSCS with 32 openMP threads on each MPI task and one MPI task on each node. Note that the 20 timings for each compiler above gave rise to identical md5sums for the `restart` file. Even for the PGI compiler. The PGI compiler version used on CSCS is the same as the one used locally, i.e. 11.10.0.

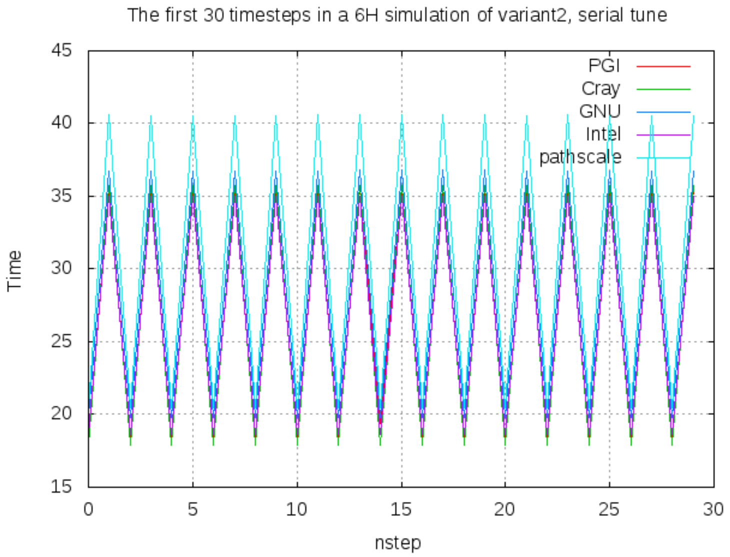


Figure 38: Serial timing of the individual time steps in the Variant2 testcase.

```
|| Time%  |       Time  | Calls   |Group
||  97.7% | 8274.659858 |       -- |      -- | 1332641.0 |USER
||------------------------------------------------------------------------------------------
||  16.6% | 1403.101359 | 0.000000 |    0.0% |   10800.0 |momeqs_wrapping_momeqs_default_
||  12.5% | 1061.241143 | 0.000000 |    0.0% |    4320.0 |tflow_tflow_int_.REGION@li.2433
||  11.9% | 1005.581515 | 0.000000 |    0.0% |    4320.0 |turbmodels_turbmodel_
||   8.3% |  700.792855 | 0.000000 |    0.0% |    4320.0 |tflow_tflow_int_.REGION@li.2473
||   7.0% |  590.279449 | 0.000000 |    0.0% |    4320.0 |tflow_tflow_int_.REGION@li.2487
||   4.6% |  393.188301 | 0.000000 |    0.0% |    4320.0 |tflow_tflow_int_.REGION@li.2500
||   4.3% |  367.693264 | 0.000000 |    0.0% |    4320.0 |tflow_tflow_int_.REGION@li.2604
||   4.2% |  357.226501 | 0.000000 |    0.0% |   10800.0 |cmod_hydrodynamics_solvemasseq_.REGION@li.564
||   4.1% |  344.334889 | 0.000000 |    0.0% |    4320.0 |tflow_tflow_int_.REGION@li.2518
||   3.8% |  318.901888 | 0.000000 |    0.0% |    4320.0 |tflow_tflow_int_.REGION@li.2456
||   3.0% |  253.305458 | 0.000000 |    0.0% |   12960.0 |dmi_mpi_dmpi_gather_copy_cmp_
||   2.4% |  204.671871 | 0.000000 |    0.0% |   10800.0 |smagorinsky_deform_
||   2.2% |  184.353608 | 0.000000 |    0.0% |   10800.0 |smagorinsky_smag_
||   1.8% |  156.418365 | 0.000000 |    0.0% |   12960.0 |dmi_mpi_dmpi_gather_copy_
||   1.8% |  150.560789 | 0.000000 |    0.0% |   10800.0 |masseqs_masseqs_solver_z_
||   1.4% |  121.364414 | 0.000000 |    0.0% |    4324.0 |thermodynamic_thermodyn_1_
||   1.3% |  106.418313 | 0.000000 |    0.0% |    4324.0 |cmod_dens_dens_
||==========================================================================================
|   2.3% |  191.383935 |       -- |      -- |  126859.0 |OMP
|   0.0% |    0.000713 | 0.000000 |    0.0% |      11.0 |PTHREAD
|   0.0% |    0.000009 |       -- |      -- |       2.0 |MPI
```

Figure 39: openmp_mpi, 12 threads, 1 task profile of a 6H simulation for Variant2, PGI, tune flag

```
|| Time%  |       Time  |   Calls   |Group
|| Time%  |      Time   |     Imb.  |  Imb.  |   Calls   |Group
||        |             |     Time  | Time%  |           | Function
||  98.1% | 1258.164507 |        -- |     -- | 1342157.9 |USER
||--------------------------------------------------------------------------------------
||  17.8% |  228.595114 | 26.694349 |  11.0% |   77768.0 |dmi_mpi_dmpi_distribute_halo_nc_
||  16.6% |  212.176394 | 34.815667 |  14.8% |   12960.0 |dmi_mpi_dmpi_gather_copy_cmp_
||  13.8% |  176.531156 | 60.221869 |  26.8% |  125284.0 |dmi_mpi_dmpi_halo_nonblocking_
||   6.8% |   87.157340 | 31.008156 |  27.6% |    8930.0 |dmi_mpi_dmpi_gather_all_
||   6.6% |   84.192399 | 41.475968 |  34.7% |   12960.0 |dmi_mpi_dmpi_gather_copy_
||   4.9% |   62.273548 |  4.495108 |   7.1% |   10800.0 |momeqs_wrapping_momeqs_default_
||   4.0% |   51.696953 |  4.025908 |   7.6% |    4320.0 |turbmodels_turbmodel_
||   3.7% |   46.811129 |  0.969382 |   2.1% |    4320.0 |tflow_tflow_int_.REGION@li.2433
||   2.2% |   28.368205 |  1.100154 |   3.9% |    4320.0 |tflow_tflow_int_.REGION@li.2473
||   2.0% |   25.078865 |  0.931486 |   3.8% |    4320.0 |tflow_tflow_int_.REGION@li.2487
||   1.5% |   19.858456 |  1.105492 |   5.6% |       6.0 |dmi_mpi_dmpi_broadcast_met_info_
||   1.3% |   16.370841 |  0.522228 |   3.3% |    4320.0 |tflow_tflow_int_.REGION@li.2604
||   1.3% |   16.170459 |  0.895640 |   5.5% |    4320.0 |tflow_tflow_int_.REGION@li.2500
||   1.2% |   15.752806 |  0.591450 |   3.8% |    4320.0 |tflow_tflow_int_.REGION@li.2518
||   1.1% |   14.653804 |  0.669864 |   4.6% |    4320.0 |tflow_tflow_int_.REGION@li.2456
||======================================================================================
|   1.3% |   16.444462 |        -- |     -- |  135362.4 |OMP
|   0.6% |    7.358511 |        -- |     -- |       2.0 |MPI
|   0.0% |    0.000761 |  0.000083 |  10.4% |      11.0 |PTHREAD
```

Figure 40: openmp_mpi, 12 threads, 20 task profile of a 6H simulation for Variant2, PGI, tune flag

9 Variant3

Table 26 summarizes the setup and figure 41 shows how the six sub-domains[22] nest to each other. The I_r number for this setup is 321.2. The as-is MPI scaling on a Cray XE6 is show in figure 42. For comparison, a 6H simulation on a X7550 xeon with 32 cores using pure openMP with the PGI compiler takes 3825.7 seconds to complete.

	IDW	**BS**	**WSNS**	**rNS**	**NA**	**MS**
resolution [n.m.]	0.25	1.0	1.0	1.0	3.0	3.0
mmx [N/S]	964	720	389	660	826	341
nmx [W/E]	792	567	208	381	321	567
kmx	77	122	110	46	78	84
gridpoints	58788576	49805280	8900320	11567160	20681388	16241148
iw2	323536	119334	32515	15593	104527	73746
iw3	6335114	6113599	1560598	409049	5648540	4214785
f_{iw3}	10.78 %	12.4%	17.8 %	31.8%	27.3%	26.0%
φ [latitude]	57 35 52.5N	65 53 30N	59 41 30N	59 29 30N	64 13 30N	47 16 30N
λ [longitude]	09 20 12.5E	14 35 50E	06 10 50E	04 09 10W	16 22 12W	5 27 30W
$\Delta\varphi$	0 0 15	0 1 00	0 1 00	0 1 00	0 3 0	0 3 0
$\Delta\lambda$	0 0 25	0 1 40	0 1 40	0 1 40	0 5 0	0 5 0
dt [sec]	5	10	10	10	10	10
maxdepth [m]	78.0	398.0	696.25	336.96	6087.69	5066.49
min Δx	413.79	1261.65	1558.76	1568.05	4029.34	5559.78
CFL	0.632	0.728	0.957	0.662	0.798	0.708
I_r	120.1	57.9	14.8	34.9	58.6	39.9

Table 26: The testcase termed Variant3. The I_r number for the setup is 321.2.

[22] Thanks to Jens Murawsky, DMI, for preparing this setup.

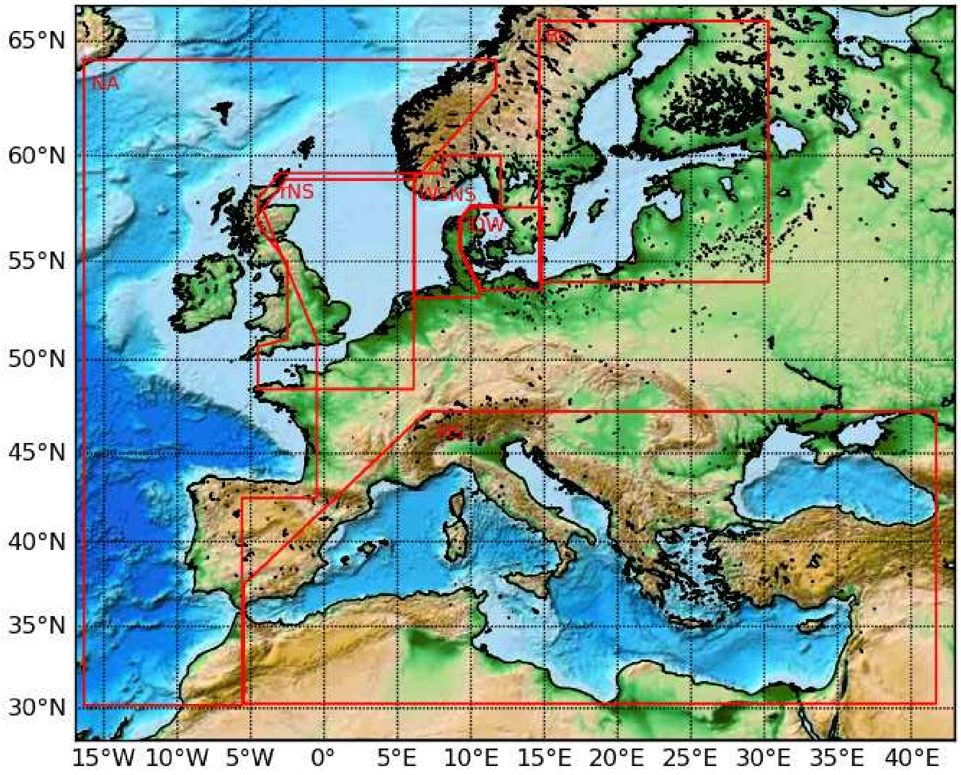


Figure 41: The Variant3 testcase

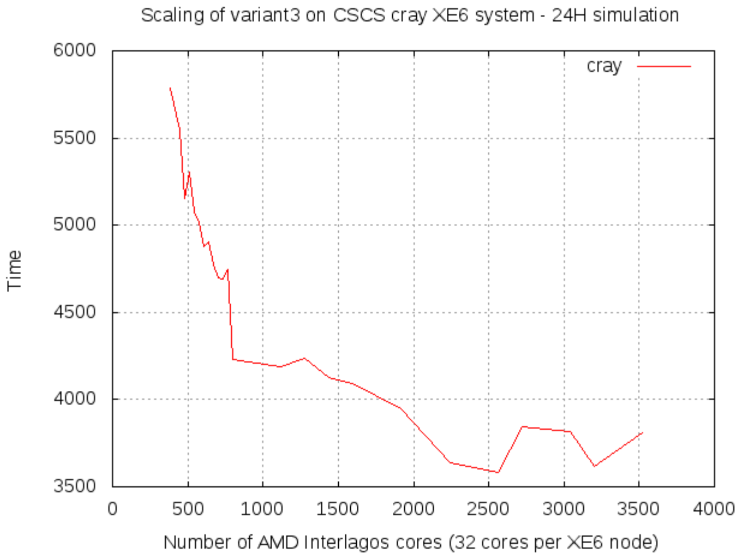


Figure 42: Scaling of the variant3 using automatically generated I-slices performed on the Cray XE6 at CSCS with 32 openMP threads on each MPI task and one MPI task on each node.

A Appendix: Vector sum using floating points

The aim of this appendix is to show how one can get very wrong results (in the absolute sense) when trying to implement algorithms that are supposed to work on floating point numbers in $\mathbb{F}_n$, even for very simple problems that have straight-forward solutions in $\mathbb{R}$.

Please note that the vector lengths (200000 and 100000, respectively) used in the the first two examples below are by no means exaggerated or large compared to the sizes we operate with in our models, on the contrary, cf. table 9 in section 5, table 21 in section 6, table 23 in section 7, table 25 in section 8, and table 26 in section 9. In the third example we will estimate the average value of a snapshot of the salinity in the IDW domain of the MyOV2.1 case in table 9.

In the first example, the program in figure 43 has three implementations of a vector sum. A simple loop over the vector summing up the element, the Fortran90 implementation of sum and finally a more complicated but more accurate implementation of the sum due to [16]. This compensated implementation is based on the error-free transformation just like the former similar algorithms due to Donald Knuth and William Kahan. You can see the outcome of the three implementations in figure 44. The analytic result is shown below and one note that while the absolute error is significant for the two naive implementations the relative error is not significant. The point here is merely that the actual implementation matters.

$$nsize/2 * 10^{12} - nsize = 100000 * 10^{12} - 100000 = 99999999999900000$$

In the second example we also stick to the simple task of doing a vector sum, but this time we use values closer to those we see in an ocean circulation model, e.g. surface salinity in Inner Danish waters, and the values are much closer to each other. Again we demonstrate using Fortran90 and its 8byte and 4byte real, but the point is that the underlying problem is representation, finite precision and rounding and therefore implementation matters; we could indeed have chosen any decent programming or scripting language, e.g. Matlab and its `double` and `single` classes. The sample program is shown in figure 45 and its outcome is shown in figure 46. It uses the naive methods from the module shown in figure 47 and the smarter methods shown in figure 48. The analytic result is 2009999.91 in $\mathbb{R}$. The naive sum is correct to 12 digits in $\mathbb{F}_{64}$, but in $\mathbb{F}_{32}$ only the two first digits are correct

```
module s
 contains
 subroutine fast2sum(a,b,s,t)
    implicit none
    real(8), intent(in)  :: a,b
    real(8), intent(out) :: s,t
    real(8) :: z
    if (b>a) then
      s=a+b
      z=s-b
      t=a-z
    else
      s=a+b
      z=s-a
      t=b-z
    endif
  end subroutine fast2sum

  subroutine compensated_sum(nsize,x,sumout)
    implicit none
    integer(4), intent(in)  :: nsize
    real(8),    intent(in)  :: x(0:)
    real(8),    intent(out) :: sumout
    real(8)    :: c, u, y, t, s, v, z
    integer(4) :: i
    s = x(1)
    c = 0.0_8
    do i=2,nsize
      call fast2sum(c,x(i),y,u)
      call fast2sum(s,y,t,v)
      z = u + v
      call fast2sum(t,z,s,c)
    enddo
    sumout = s
  end subroutine compensated_sum
end module s

program testsum
  use s
  implicit none
  integer(4), parameter :: nsize=200000
  real(8) ::asum(nsize),x
  integer(4) :: i
  do i=1,nsize,2
    asum(i)=-1.0_8
  enddo
  do i=1,nsize-1,2
    asum(i+1)=1000000000000_8
  enddo
  ! naive sum
  x=0
  do i=1,nsize
    x=x+asum(i)
  enddo
  write (*,*) 'Naive sum ', x, ' error is ', x-99999999999900000_8
  write (*,*) 'Fortran sum ', sum(asum(1:nsize)), ' error is ', sum(asum(1:nsize))-99999999999900000_8
  ! compensated sum
  call compensated_sum(nsize,asum,x)
  write (*,*) 'Compensated sum ', x, ' error is ', 99999999999900000_8-x
end
```

Figure 43: Three implementations of a vector sum of floating point numbers

```
jwp@munin-2:~> ./a.out
 Naive sum    9.9999999999990992E+016  error is    90992.0000000000
 Fortran sum    9.9999999999990992E+016  error is    90992.0000000000
 Compensated sum    9.9999999999900000E+016  error is  0.00000000000000
jwp@munin-2:~>
```

Figure 44: Testing the three implementations of the vector sum.

and the result is so bad that an average value estimated from this value of the sum is quite a lot outside the interval from minimum to maximum value of the vector element values. It helps if the summation operations of the 4byte vector elements are performed in the double precision and the result is rounded back to $\mathbb{F}_{32}$, then the result gets as good as it can be in $\mathbb{F}_{32}$. If the sums are found using the compensated sum algorithm we get the exact result in $\mathbb{F}_{64}$ and the best possible rounded result in $\mathbb{F}_{32}$.

```
program sum_examples

  use sumsubs, only : naive_sum, smart_sum
  implicit none

  integer(4), parameter   :: nsize = 100000
  real(8),    allocatable :: v64(:)
  real(8)                 :: sum64
  real(8),    parameter   :: v1_64 = 20.01_8, v2_64 = 20.1_8
  real(4),    allocatable :: v32(:)
  real(4)                 :: sum32
  real(4),    parameter   :: v1_32 = 20.01_4, v2_32 = 20.1_4

  allocate( v64(nsize), v32(nsize) )

  ! initialize:
  v64(1)       = v1_64
  v64(2:nsize) = v2_64
  v32(1)       = v1_32
  v32(2:nsize) = v2_32

  print*, 'Analytic result in R: 100000*20.1 - 0.1 + 0.01 = 2009999.91'

  call naive_sum(nsize,v64,sum64)
  write(*,*) 'Naive sum in F_64:               ', sum64

  call naive_sum(nsize,v32,sum32)
  write(*,*) 'Naive sum native in F_32:        ', sum32

  call naive_sum(nsize,v32,sum64)
  write(*,*) 'Naive sum typecasted and rounded:', real(sum64,4)

  call smart_sum(nsize,v64,sum64)
  write(*,*) 'Smart sum in F_64:               ', sum64

  call smart_sum(nsize,v32,sum32)
  write(*,*) 'Smart sum in F_32:               ', sum32

end program sum_examples
```

Figure 45: Sample code for testing different vector sums in $\mathbb{F}_{64}$ and $\mathbb{F}_{32}$.

In the third example we use the salinity in the IDW domain from an arbitrary restart file and we will try to estimate the average value. The vector length is 1583550. If we base the average value on naive sums from surface to bottom we get:

15.0156884190895 in $\mathbb{F}_{64}$
14.9965305 in $\mathbb{F}_{32}$

```
pirate:~/src/floatingpoints> gfortran -c -o smartsums.o smartsums.f90
pirate:~/src/floatingpoints> gfortran -c -o naivesums.o naivesums.f90
pirate:~/src/floatingpoints> gfortran -c -o sum_examples.o sum_examples.f90
pirate:~/src/floatingpoints> gfortran -o sum_examples sum_examples.o naivesums.o smartsums.o
pirate:~/src/floatingpoints> ./sum_examples
 Analytic result in R: 100000*20.1 - 0.1 + 0.01 = 2009999.91
 Naive sum in F_64:                  2009999.91000360
 Naive sum native in F_32:           2011733.
 Naive sum typecasted and rounded:   2010000.
 Smart sum in F_64:                  2009999.91000000
 Smart sum in F_32:                  2010000.
```

Figure 46: Testing different implementations of the vector sum in $\mathbb{F}_{64}$ and $\mathbb{F}_{32}$.

```
module naivesums
  implicit none

  interface naive_sum
    module procedure naive_sum_64
    module procedure naive_sum_32_native
    module procedure naive_sum_32_typecasted
  end interface

  private :: naive_sum_64, naive_sum_32_native, naive_sum_32_typecasted
  public  :: naive_sum

contains

  subroutine naive_sum_64( n, x, sumout )
    integer(4), intent(in)  :: n
    real(8),    intent(in)  :: x(:)
    real(8),    intent(out) :: sumout
    integer(4) :: i
    sumout = 0.0_8
    do i=1,n
      sumout = sumout + x(i)
    enddo
  end subroutine naive_sum_64

  subroutine naive_sum_32_native( n, x, sumout )
    integer(4), intent(in)  :: n
    real(4),    intent(in)  :: x(:)
    real(4),    intent(out) :: sumout
    integer(4) :: i
    sumout = 0.0_4
    do i=1,n
      sumout = sumout + x(i)
    enddo
  end subroutine naive_sum_32_native

  subroutine naive_sum_32_typecasted( n, x, sumout )
    integer(4), intent(in)  :: n
    real(4),    intent(in)  :: x(:)
    real(8),    intent(out) :: sumout
    integer(4) :: i
    sumout = 0.0_8
    do i=1,n
      sumout = sumout + real(x(i),8)
    enddo
  end subroutine naive_sum_32_typecasted

end module naivesums
```

Figure 47: Naive vector sum implementations in $\mathbb{F}_{64}$ and $\mathbb{F}_{32}$.

```
module smartsums
  implicit none
  interface smart_sum
    module procedure priest_sum_64
    module procedure priest_sum_32
  end interface
  private :: priest_sum_64, priest_sum_32, fast2sum_64, fast2sum_32
  public  :: smart_sum
contains
  subroutine priest_sum_64( n, x, sumout )
    implicit none
    integer(4), intent(in)  :: n
    real(8),    intent(in)  :: x(:)
    real(8),    intent(out) :: sumout
    real(8)    :: c, u, y, t, s, v, z
    integer(4) :: i
    s = x(1)
    c = 0.0_8
    do i=2,n
      call fast2sum_64(c,x(i),y,u)
      call fast2sum_64(s,y,t,v)
      z = u + v
      call fast2sum_64(t,z,s,c)
    enddo
    sumout = s
  end subroutine priest_sum_64

  subroutine fast2sum_64(a,b,s,t)
    implicit none
    real(8), intent(in)  :: a,b
    real(8), intent(out) :: s,t
    real(8) :: z
    if (b>a) then
      s=a+b
      z=s-b
      t=a-z
    else
      s=a+b
      z=s-a
      t=b-z
    endif
  end subroutine fast2sum_64

  subroutine priest_sum_32( n, x, sumout )
    implicit none
    integer(4), intent(in)  :: n
    real(4),    intent(in)  :: x(:)
    real(4),    intent(out) :: sumout
    real(4)    :: c, u, y, t, s, v, z
    integer(4) :: i
    s = x(1)
    c = 0.0_4
    do i=2,n
      call fast2sum_32(c,x(i),y,u)
      call fast2sum_32(s,y,t,v)
      z = u + v
      call fast2sum_32(t,z,s,c)
    enddo
    sumout = s
  end subroutine priest_sum_32

  subroutine fast2sum_32(a,b,s,t)
    implicit none
    real(4), intent(in)  :: a,b
    real(4), intent(out) :: s,t
    real(4) :: z
    if (b>a) then
      s=a+b
      z=s-b
      t=a-z
    else
      s=a+b
      z=s-a
      t=b-z
    endif
  end subroutine fast2sum_32
end module smartsums
```

Figure 48: Compensated vector sum implementations in $\mathbb{F}_{64}$ and $\mathbb{F}_{32}$.

and if we do the naive sum from the bottom and up we get:
`15.0156884190889` in $\mathbb{F}_{64}$
`15.0420637` in $\mathbb{F}_{32}$

For comparison, if we base the estimate on the compensated sum algorithm we get:

`15.0156884190895` in $\mathbb{F}_{64}$

That is, the naive sum is quite sensitive to permutation of the data and in $\mathbb{F}_{32}$ we may obtain as little as 1 or 3 correct digits while we have 12-13 correct digits in $\mathbb{F}_{64}$.

The above are just examples that demonstrate that one should think twice about the implementation; it is not always as straight-forward as one could hope. The reader is encouraged to play with more examples of estimating the sum or the average value of different data sets; try e.g. random numbers and vector lengths of 100.000, 1.000.000 and 10.000.000 and collect findings similar to those above. And it's not always the vector length that counts; try to add the following two numbers[23]

```
v1 = 9876543210.20000
v2 = -9876543210.10000
```

in $\mathbb{F}_{64}$.

[23] This last example is attributed to Peter Sestoft, Professor, IT University of Copenhagen, Denmark, and he shows more nasty and interesting examples in:
`http://www.itu.dk/courses/KF04/F2010/uge4/computernumbers.pdf`

B Appendix: Sea-level maps

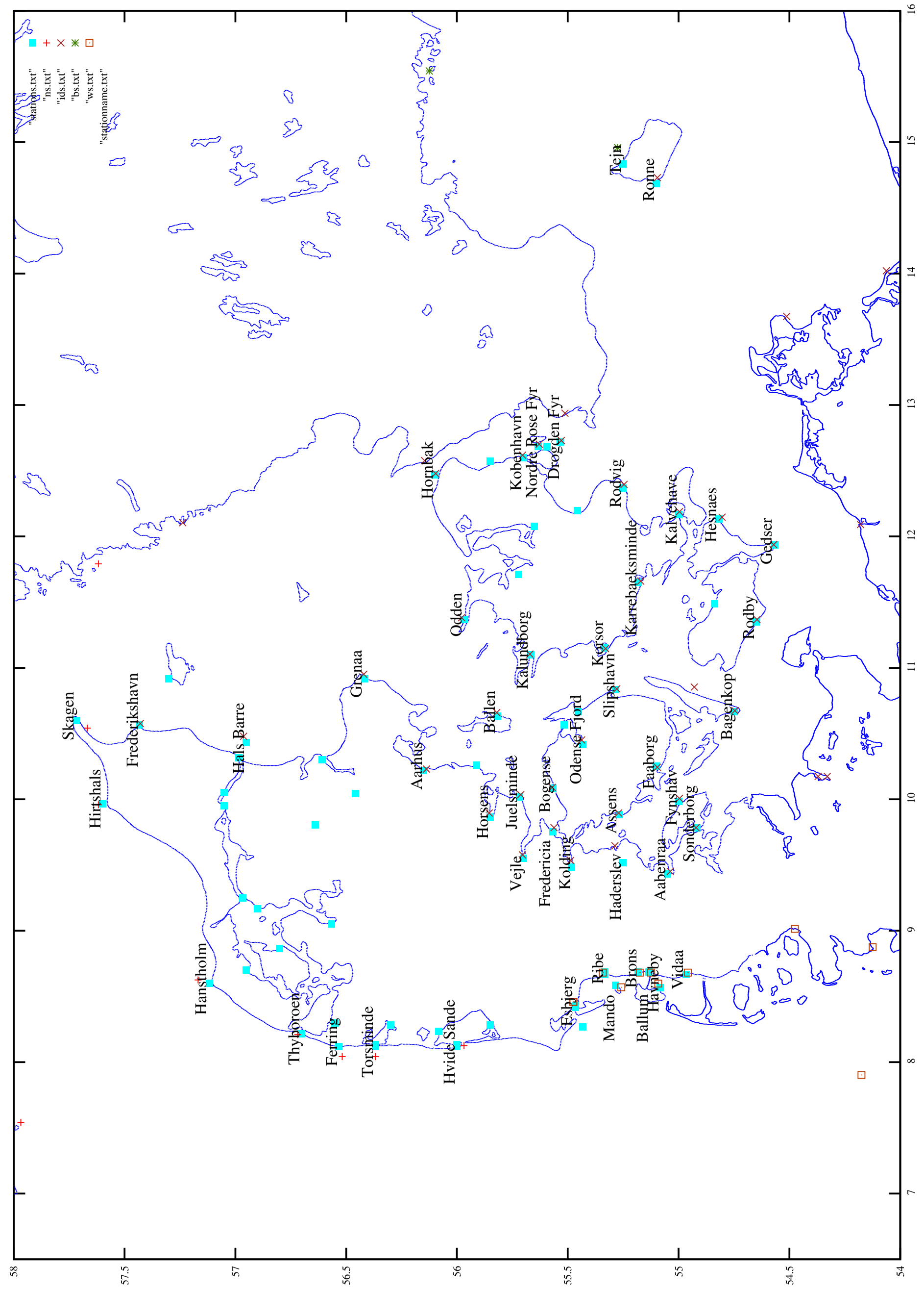


Figure 49: Map showing names of the Danish stations and the placements where the model produces sea-levels every 10 minute.

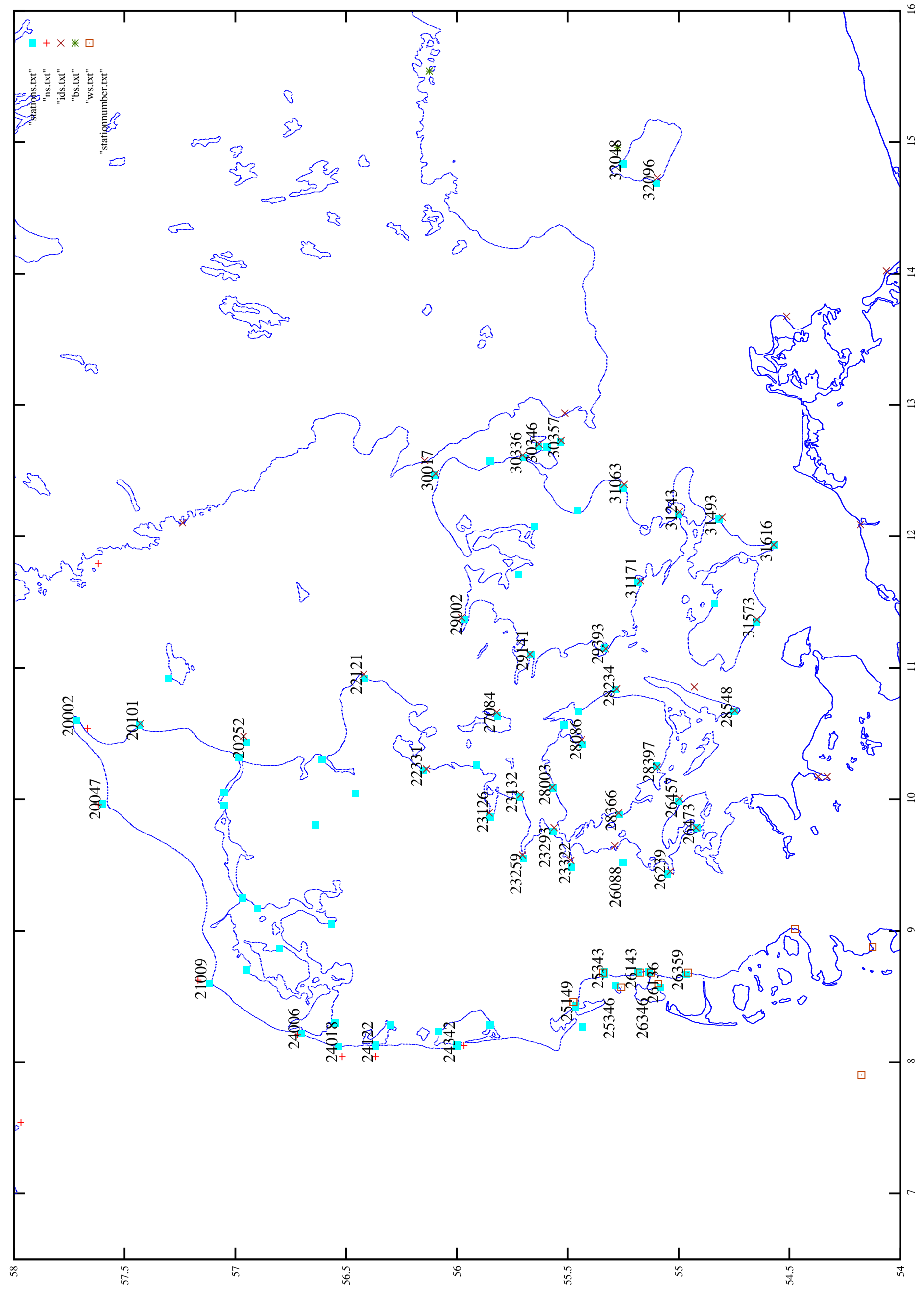


Figure 50: Map showing numbers of the Danish stations and the placements where the model produces sea-levels every 10 minute.

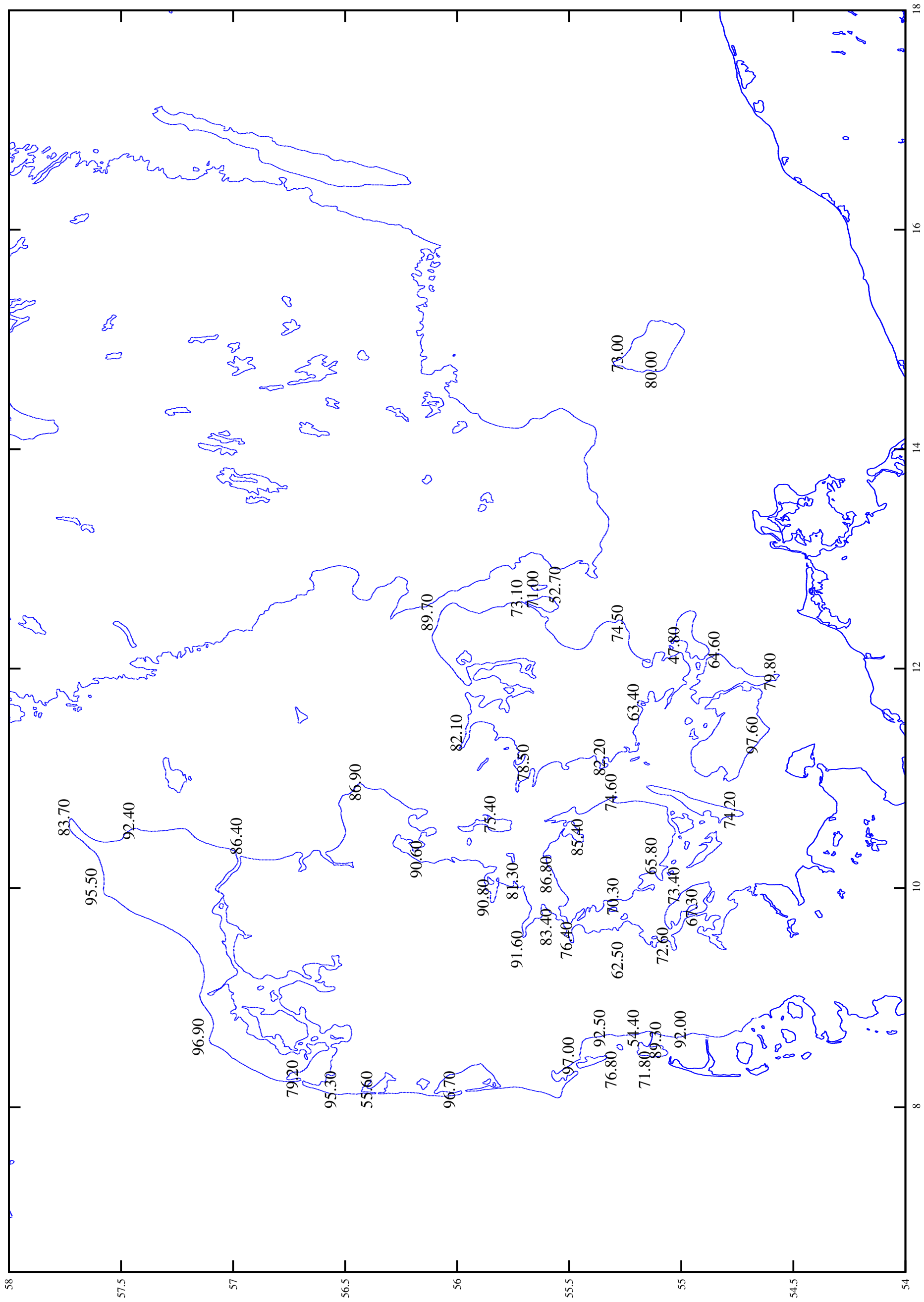


Figure 51: Map showing observation coverage for each of the Danish stations in 2011 with the described screening procedure.

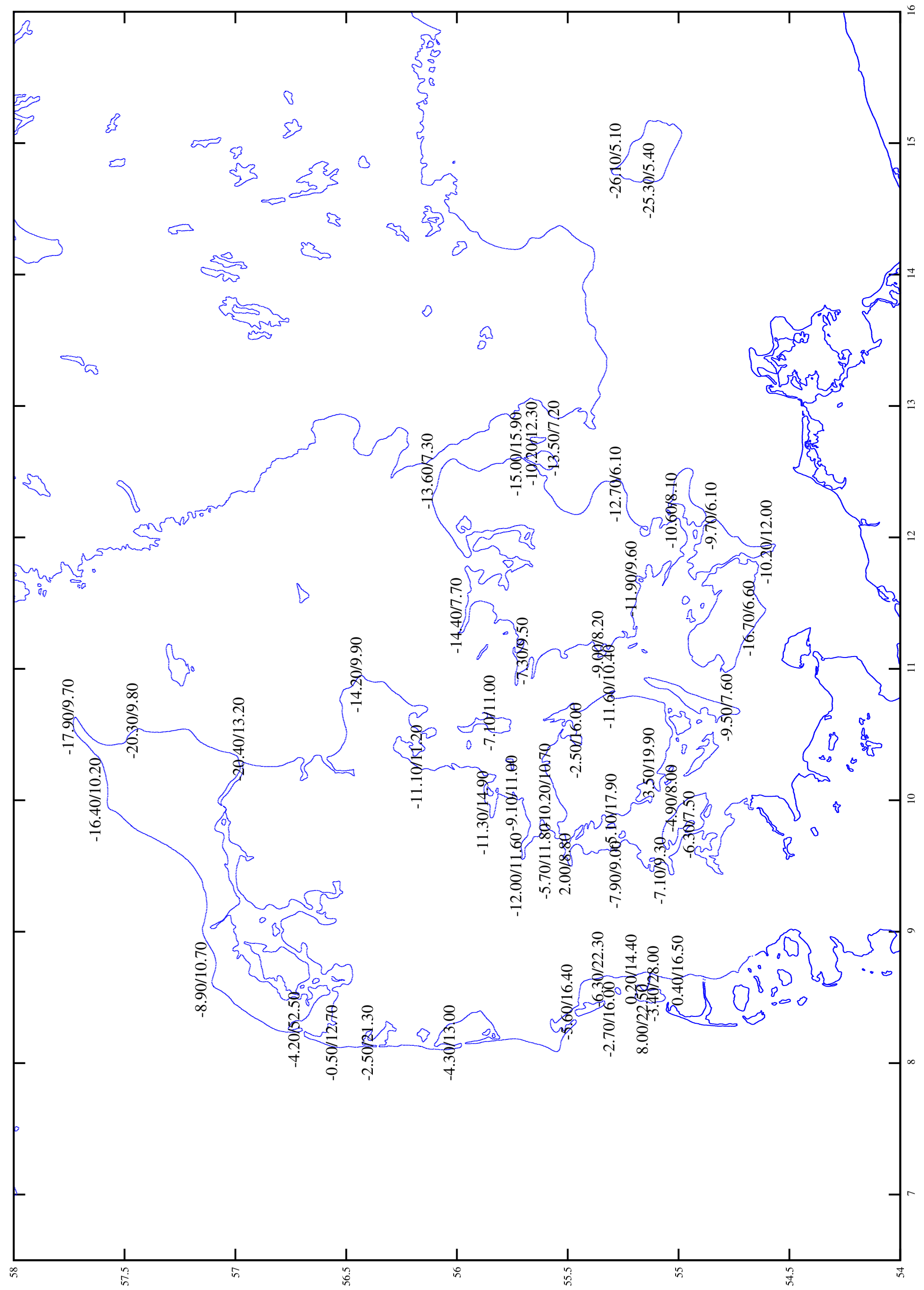


Figure 52: Map showing verification scores in terms of bias/std for each of the Danish stations in 2011.

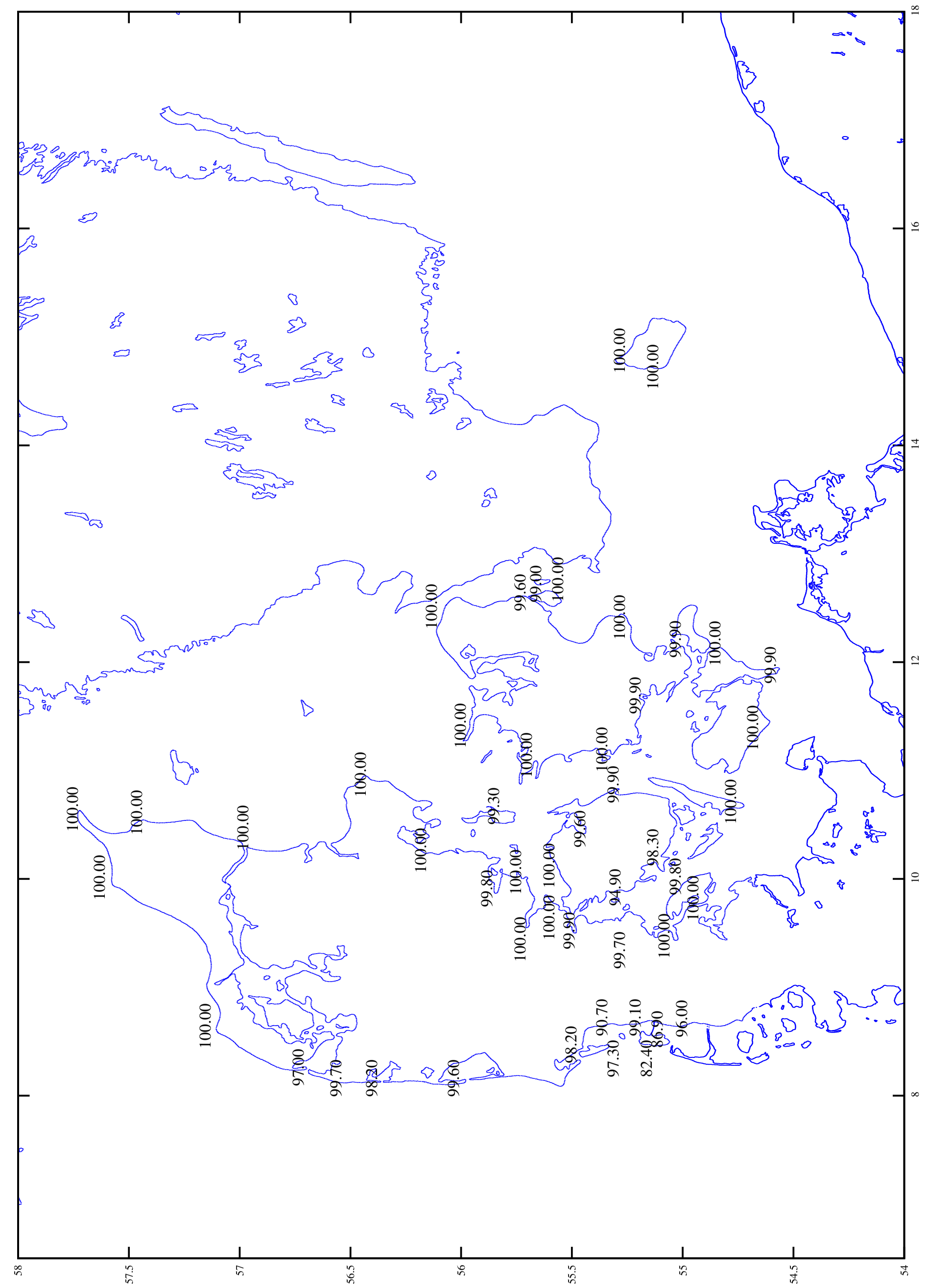


Figure 53: Map showing verification scores in terms of hit-rates. The model hits the observation if the absolute difference between the two is less than 20 cm.

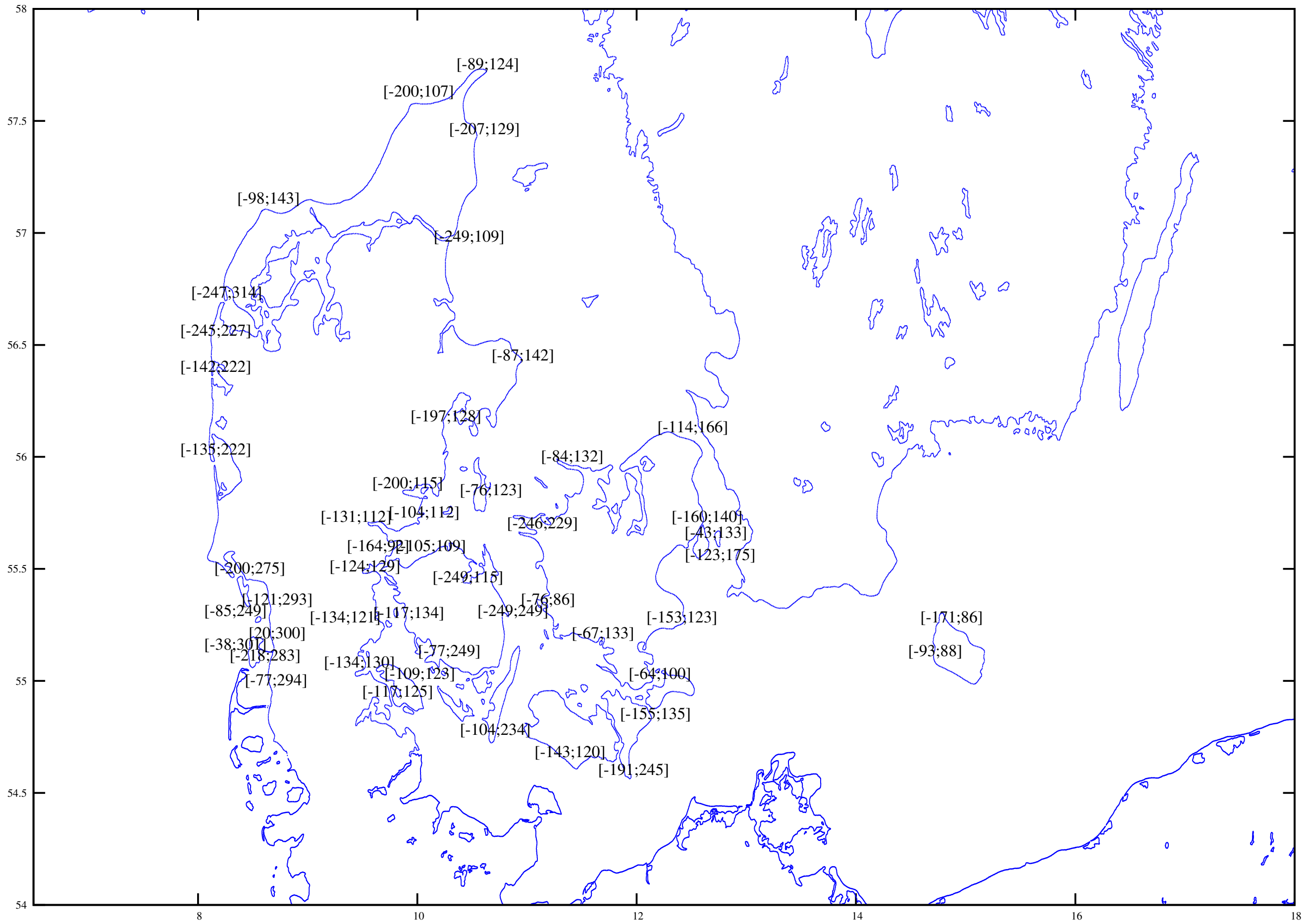


Figure 54: Map showing the sea-level range measured at the Danish stations throughout 2011.

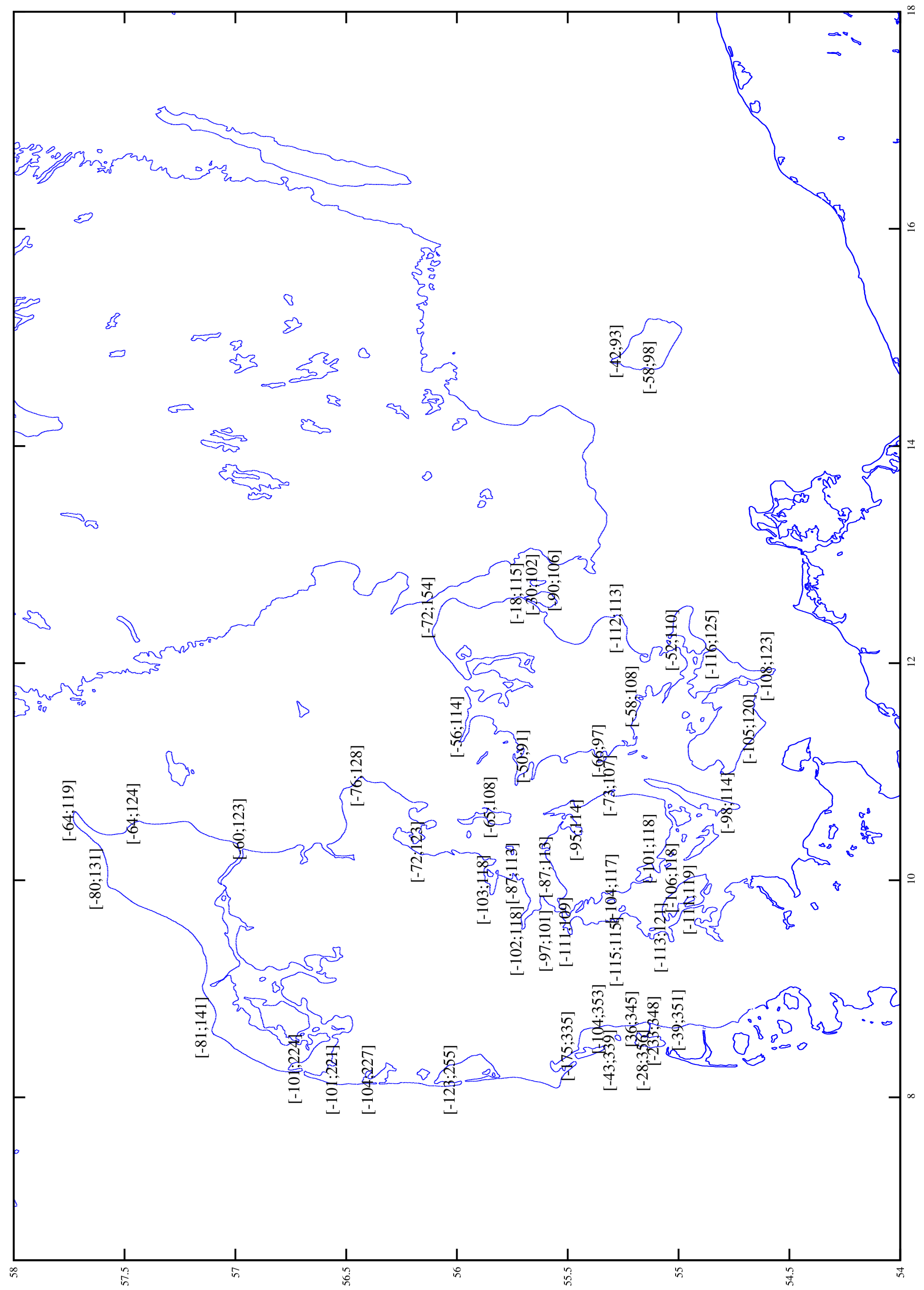


Figure 55: Map showing the sea-level range obtained by the model throughout 2011.

C Appendix: Abbrevations.

cc-NUMA Cache-Coherent Non-Uniform Memory Access.

CFL Courant, Friedrichs and Levy; the CFL-condition is named after Richard Courant, Kurt Friedrichs, and Hans Lewy who described it in their 1928 paper.

CPU Central Processing Unit.

CSCS Centro Svizzero di Calcolo Scientifico; the Swiss National Supercomputing Centre.

DMI Danish Meteorological Institute.

EPS Ensemble Prediction System.

$\mathbb{F}$, $\mathbb{F}_n$ The set of floating point numbers in arbitrary precision and in n-bit precision.

FDS Finite Difference Scheme.

FMA Fused Multiply-Add; a floating-point multiply-and-add operation performed in one step, with a single rounding.

FTCS Forward-in-Time, Central-in-Space; refers to the centering of a finite difference scheme.

GPU Graphics Processing Unit.

HBM HIROMB-BOOS Model.

IBVP Initial Boundary Value Problem.

IEEE Institute of Electrical and Electronics Engineers; the world's largest professional association for the advancement of technology.

IO Input/Output.

MPI Message Passing Interface; a standardized and portable message-passing system designed to function on a wide variety of parallel computers.

MyO v2.1 The version 2.1 of the operational forecast model run under the MyOcean project.

n.m. Nautical Miles,; 1 n.m. is approximately 1852 meters.

openACC A programming standard for parallel computing, designed to simplify parallel programming of heterogeneous CPU/GPU systems.

openMP Open MultiProcessing; an application programming interface that supports multi-platform shared memory multiprocessing programming in languages such as Fortran and C.

PDE Partial Differential Equation.

PP Physical Problem.

$\mathbb{R}$ The set of real numbers in mathematics.

SSE Streaming SIMD Extensions; one of the Intel SIMD (Single Instruction, Multiple Data) processor supplementary instruction sets.

TVD scheme Total Variation Diminishing scheme; is a property of certain discretization schemes used to solve hyperbolic partial differential equations; A TVD scheme is monotonicity preserving.

1D, 2D and 3D one-, two- and three-dimensional; refers to the number of spatial dimensions at hand.

References

[1] M. B. Abbott and D. R. Basco. *Computational fluid dynamics: An introduction for engineers*. Longman, 1989.

[2] Per Berg. Mixing in HBM. DMI Scientific Report No. 12-03. Technical report, DMI, Copenhagen, 2012.

[3] Per Berg and Jacob Weismann Poulsen. Implementation details for HBM. DMI Technical Report No. 12-11. Technical report, DMI, Copenhagen, 2012.

[4] George F. Carrier and Carl E. Pearson. *Partial Differential Equations, Theory and Techniques*. Academic Press, 1976.

[5] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, March 1991.

[6] Nicholas J. Higham. Bounding the error in gaussian elimination for tridiagonal systems. *SIAM J. Matrix Anal. Appl*, 11:521–530, 1990.

[7] Ph. Langlois, N. Louvet Mai, Ph. Langlois, and N. Louvet. Solving triangular systems more accurately and efficiently, 2005.

[8] A.D. McCowan, E.B. Rasmussen, and P. Berg. Improving the performance of a two-dimensional hydraulic model for floodplain applications. *The Institution of Engineers, Australia, Hobart*, 2001.

[9] David Monniaux. The pitfalls of verifying floating-point computations. *ACM Trans. Program. Lang. Syst.*, 30(3):12:1–12:41, May 2008.

[10] Jean-Michel Muller, Nicolas Brisebarre, Florent de Dinechin, Claude-Pierre Jeannerod, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, and Serge Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010. ACM G.1.0; G.1.2; G.4; B.2.0; B.2.4; F.2.1., ISBN 978-0-8176-4704-9.

[11] Günther Nausc, Rainer Feistel, and Volker Mohrhol. Water exchange between the Baltic Sea and the North Sea, and conditions in the deep basins. HELCOM Indicator Fact Sheets 2011. Online. Technical report, HELCOM, 2011.

[12] Dora B. Heras Pablo Quesada-Barriuso, Julin Lamas-Rodrguez. Selecting the best tridiagonal system solver projected on multi-core CPU and GPU platforms. volume II, pages 839–845, Las Vegas (USA), 18/07/2011 2011.

[13] Velisar Pavlov and Plamen Y. Yalamov. Stability issues of the Wang's partitioning algorithm for banded and tridiagonal linear systems. In *Euro-Par'99*, pages 1149–1152, 1999.

[14] G. P. Pedersen. *Analysis Now*, volume 118 of *Graduate Texts in Mathematics*. Springer, 1989.

[15] William H. Press, Saul A. Teukolsky, William T. Vettering, and Brian P. Flannery. *Numerical Recipes, The Art of Scientific Computing, Second Edition*. Cambridge University Press, 1992.

[16] Douglas M. Priest. On properties of floating point arithmetics: Numerical stability and the cost of accurate computations. Technical report, University of California at Berkeley, 1992.

[17] H. H. Wang. A parallel method for tridiagonal equations. *ACM Transactions on Mathematical Software*, pages 170–183, 1981.

[18] Yao Zhang, Jonathan Cohen, and John D. Owens. Fast tridiagonal solvers on the GPU. In *PPOPP*, pages 127–136, 2010.

Previous reports
Previous reports from the Danish Meteorological Institute can be found on:
http://www.dmi.dk/dmi/dmi-publikationer.htm