Inštitut Jožef Stefan
1000 Ljubljana, Jamova 39
Odsek E6 - KOMUNIKACIJSKI SISTEMI

IJS-DP-11167

Analiza in optimizacija implementacije modela NAPOM za Morsko biološko Postajo.

M. Depolli, J. Ugovšek, R. Trobec, G. Kosec

December 2012

Analysis and optimization of NAPOM implementation

M. Depolli, J. Ugovšek, R. Trobec, G. Kosec
Jožef Stefan Institute, Department of Communication Systems – E6, Jamova 39 1000 Ljubljana, Slovenia

Abstract
In this project report, the main features of the current NAPOM program code implementation and its execution performance on the Marine Biology Station (MBS) computing cluster are assessed. The FORTRAN source code is examined and analysed to determine the behaviour of the numerical implementation of the physical model, while the hardware architecture MBS cluster is tested to design the most effective optimization and parallelization action. Bottlenecks are identified on both ends. The improvements of the main program code as well as the redesign of the pre-process scripts are performed in order to achieve shorter execution time. Most modules of the NAPOM package are optimized to achieve maximal performance regarding the hardware architecture, specifically memory architecture. The pre-process modules are distributed on more computational nodes while all independent complex operations are parallelized with the shared memory principles. The optimized/parallelized implementation of a NAPOM package executes four times faster than the original one with only a minimal additional load to the MBS cluster, i.e. with the same consumption of energy.

Keywords:
NAPOM, FORTRAN, cache, OpenMP, parallelization, optimization

1 Introduction

The Princeton Ocean Model (POM) [1] is a 3D primitive formulation model. The model considers fluid flow through Navier Stokes momentum equation coupled with non-linear equations of state together with energy and salinity transport. POM uses two level Euler temporal discretization and Finite Differences Method (FDM) [2] for spatial discretization. The horizontal velocity components, heat, salinity and kinetic energy transport are solved explicitly (internal mode), while pressure, density and vertical velocity component are solved implicitly (external mode). The internal and external modes are solved at different temporal resolutions. The splitting rate is set based on the integration stability criteria (CFL conditions) [3]. The dynamics of the internal mode is much less intense and thus it uses longer time step in comparison with the external mode. The internal mode is computed on a 3D domain, while the external considers a 2D domain.

The POM model stands for standard numerical tool for circulation forecast in several countries. The augmented variant of POM, referred as NAPOM (North Adriatic POM) is used for academic and operative forecast at Marine Biology Station in Piran as well as at the Slovenian Environment Agency (SEA). The NAPOM is derived from a variant of POM, which undergone last changes in 2006. The NAPOM incorporates local bathymetry, products form local meteorological models (boundary and initial conditions), and daily measurements of rivers discharge and temperature. The whole numerical package consists of two major parts; first, the pre-process where all the required data is gathered and adequately transformed, and second, the main code, where numerical simulation takes place. The original POM is written in FORTRAN 77; however, the augmentations are written in a mixture of FORTRAN 77 and FORTRAN 90. The code uses single precision data type and is assembled mainly in a single source file. The current code is sequential. The main goal of the present work is to analyse the computational performance of the NAPOM, identify flaws and propose improvements. Besides the optimization of the code, the parallelization is to be considered, as well.

2 Current status

The project starts with the analysis of the current run-time environment status. The work is performed directly on a copy of the operative code installed on the MBS cluster. The task is divided in three main parts

- analysis of the pre-process procedures,
- analysis of the main program code and
- analysis of the cluster hardware that executes the main code.

2.1. Pre-process procedures

During the pre-process various scripts initiate automatic connections to the SEA and the INGV servers and download data required to set up the simulation environment. Few hundreds of Mb of data is transferred and processed before each simulation run. The whole pre-process procedure is executed sequentially on the head node, where the processed data is stored in several binary files. The main pre-process steps together with the current average execution time are presented in Table I.

2.2. Main program code

After the data preparation in the pre-process procedures, the main NAPOM code is executed. The code is compiled with the PGI FORTRAN compiler [4], which has been upgraded from version 7.2 to version 12.8 within the current optimization task. The NAPOM runs with a time step of 90 s on an orthogonal cell grid of 600x600 m with 10 vertical layers. The simulation covers a period of three days (real time) in 2880 iterations. Maximal velocities are within the range of 1 m/s and, according to the POM manual [5], the stability criteria are satisfied. The modules of the main code, together with the average execution time, are presented in Table II and in the graphical form with a bar plot in Figure 1.

Table I: The pre-process elements

#	**script**	**Description**	**execution time [s]**
1	get.arso.data.sh	Retrieve 140Mb of data from SEA.	14.20
2	get.ingv.data.sh	Retrieve 224 Mb of data from INGV server.	78.40
3	extract.arso.data.sh	Extracting the data from SEA.	88.05
4	run.cmsj2napom.sh	Preparation of SEA data for NAPOM.	398.20
5	init_ingv_ts_obc.sh	Preparation of INGV data for NAPOM.	144.10
6	run.clima.sh		4.00
		cumulative pre-process time	726.95

Table II: The NAPOM main code elements

#	Description	execution time 1 [s]
7	program initialization	0.47
8	initial outputs	0.09
9	internal loop initialization	11.62
10	computations of lateral viscosity	209.24
11	external loop initialization	14.26
12	boundary conditions(1) & advave	77.00
13	compute ua, va	62.46
14	boundary conditions (2)	16.91
15	8000 final loops	20.97
16	vertvl & boundary conditions (5)	30.41
17	advq called on T and S, profq & boundary conditions (6)	313.86
18	advt called on T and S	600.26
19	proft called on T and S, & boundary conditions (4)	128.40
20	dens	57.87
21	compute uf, vf & boundary conditions (3)	144.06
22	output routines at the 9000 end	7.72
	Cumulative execution time [s]	1695.60

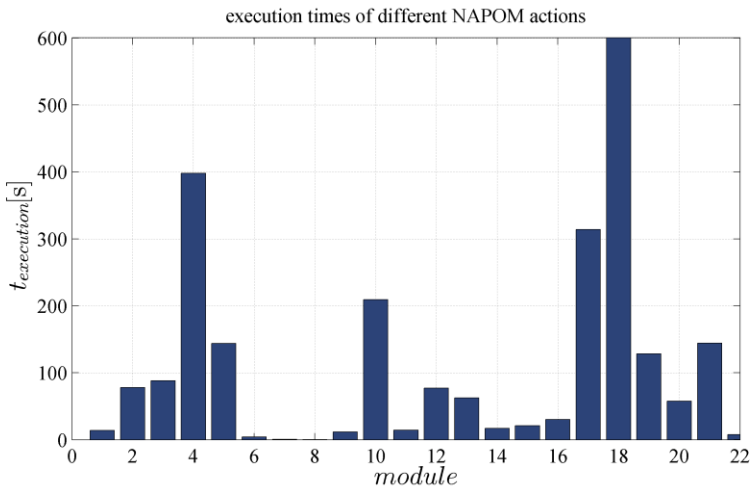

Figure 1: Execution time of different NAPOM modules

2.3. Hardware setup

In order to start an effective optimization, the computation resources have to be adequately analysed. Besides the CPU power, the memory architecture also plays an important role in

intense computations. From the distributed parallelization point of view, the network performance is equally important, as well.

The MBS uses a computer cluster built from five computing nodes. Each node has a single 4-core Intel(R) Xeon(R) CPU E5440 @ 2.83GHz with 64 kB of L1 cache per core. Two pairs of cores share 6 MB of L2 caches and all cores share 8 GB of the main memory. Two Gigabit Ethernet ports are bonded in a load balancing (round-robin) mode and connected to a switch. The nodes are running Linux Ubuntu 8.04.2 operating system.

For better presentation, all measured results obtained from MBS computing cluster are compared against the reference test cluster, installed at IJS-E6, The reference cluster is built from 36 nodes, each with Intel(R) Xeon(R) CPU E5520 @ 2.27GHz with 64 kB of L1 and 256 kB of L2 cache per core, and a shared 8 MB L3 cache. Each node also comprises 6 GB of 1066 MHZ DDR3 RAM, connected in a three channel configuration and 8 Gigabit Ethernet ports. All 288 Ethernet ports are connected to 8 40-port Gigabit Ethernet switches for a reconfigurable interconnection network of the cluster. The nodes are running Ubuntu 12.04 operating system.

The first measurement is focused on the **speed of floating point** computations. The set of basic floating point operations, and several combined operations, are used in the benchmarking (Table III).

Table III: Function list with 32 bit float variables a, b, c;

#	Function	#	Function	#	Function	#	Function
1	a	**7**	c=a-b	**13**	c=pow(a,b)	**20**	c=min(a,b)
2	c=0	**8**	c=a*M_PI	**14**	c=exp(a)	**21**	c=exp(a)+exp(b)
3	c=M_PI	**9**	c=a*b	**15**	c=log2(a)	**22**	c=sin(a)+cos(a)
4	c=a	**10**	c=a/b	**16**	c=sin(a)	**23**	function(c,a,b)
5	c=a+M_PI	**11**	c=a%b	**17**	c=atan2(a)	**24**	c=rand [0..1]
6	c=a+b	**12**	c=(a+b)*(a-b)*(b-a)+b	**18**	c=sqrt(a)		

The results are stated in averaged time (over 10^4 repetitions) needed to compute each operation. The comparison of results between the reference and MBS clusters is presented in Figure 2. Note that the performances of both CPUs are similar. More complex operations are computed faster on the reference machine, while some simpler operations are faster on the MBS machine; but, in general, the performances are within the same range.

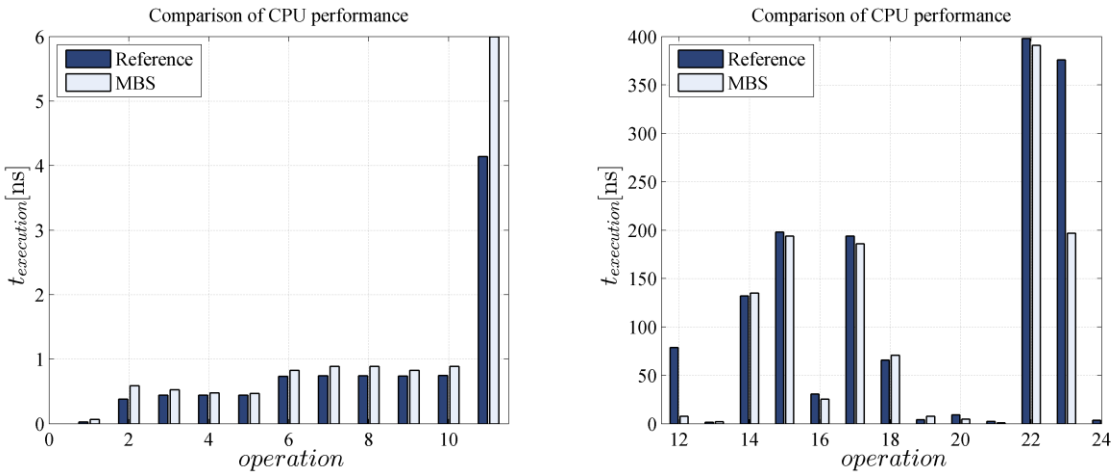

Figure 2: Comparison of CPU performance on different operations (Table III)

The **performance of memory** is tested with random reads and writes of data blocks. The size of data varies from a few kB to 64 MB. The results are presented in Figure 3. The cache effects are clearly visible; as long as the data size is small enough to fit into the cache memories, the performance of MBS nodes is sound; however, the MBS nodes suffer from the slow main memory access that is about four times slower in comparison to the reference machine.

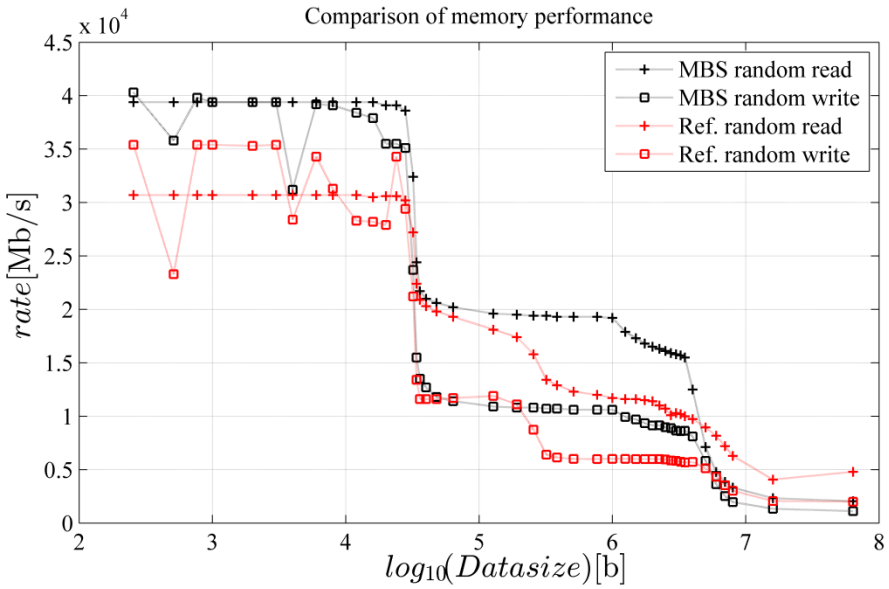

Figure 3: Comparison of memory performance.

Finally, the performance of **communication network** is tested. The results are presented in Figure 4. Again, the results are compared against the reference cluster. It is evident that the communication latencies (rate by shorter messages) are within the same range on both systems,

however, the bandwidth of the MBS interconnect is approximately two times smaller in comparison to the reference cluster.

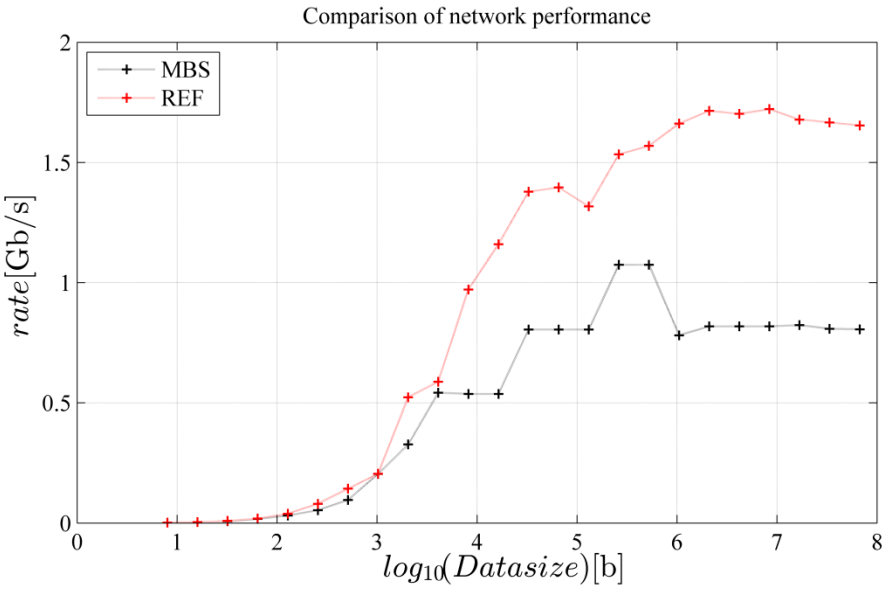

Figure 4: Comparison of network performance.

2.4. Behaviour of the numerical integration

The NAPOM model consists of the highly coupled and non-linear problems (fluid flow coupled with energy and salinity transport, turbulence models, etc.) as well as hyperbolic nature from advective dominated salinity transport. Consequently, the numerical integration of the model might be unstable, i.e. small perturbations tend to cause large differences in the final fields. The problem could be even more pronounced as the code is implemented in a single precision mode. We noticed that even changing the compiler and/or compiler flags influences the results. Similar effects are noticeable when the code is vectorized or when minor changes without any mathematical meaning, e.g. various numerical operation sequences, are inserted. To confirm that the enumerated effects are a consequence of rounding errors, simple analysis is done. The input data is perturbed just on the last bit, where the rounding takes place. The results obtained from the perturbed input are compared against the results obtained from the non-perturbed input. The differences are expected to be minimal. The difference is calculated, for each velocity layer, as an absolute difference between the original and the perturbed fields, normalized with the maximum value within the considered layer. From Table IV one can see that the maximal differences are substantial (3 to 9 %) in all layers, which indicates that some ill-conditioned modules could be present within the current NAPOM code. In Figure 5 the results of the original and the perturbed case together with the difference for *w* velocity component on layer 4 are shown.

Table IV: Difference statistics for different w-layer.

layer (z)	max (difference)	mean (difference)	std (difference)
1	0.00e+00	0.00e+00	0.00e+00
2	8.68e-02	5.18e-04	1.30e-03
3	6.75e-02	3.85e-04	9.22e-04
4	6.97e-02	3.24e-04	7.96e-04
5	5.38e-02	2.74e-04	6.56e-04
6	3.79e-02	2.48e-04	5.48e-04
7	3.72e-02	2.45e-04	5.54e-04
8	3.59e-02	2.50e-04	5.51e-04
9	2.74e-02	2.55e-04	5.61e-04
10	8.47e-02	2.76e-04	9.88e-04
11	8.47e-02	2.76e-04	9.88e-04

The behaviour with the respect to other actions, e.g. changing compiler, flags, coding approach, etc., have also been tested. All actions introduce a considerable change (difference) in the final result. The difference tolerance for further testing and comparisons of obtained results is set to the range obtained from the presented test in Table IV. Although the topic should be further investigated, the discussions about the stability, and other related topics are out of the scope of the current work.

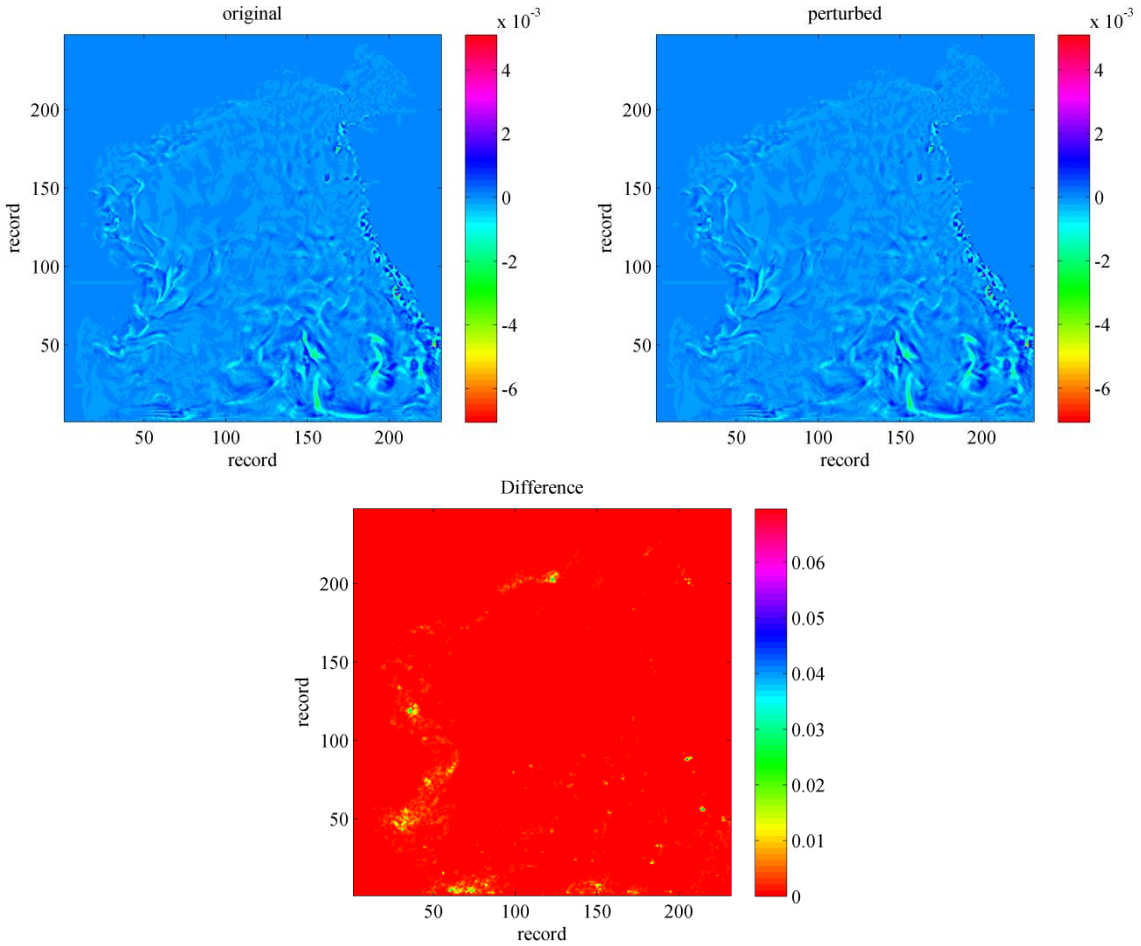

Figure 5: Comparison of original and perturbed calculation and difference between them.

3 Optimization strategy

Based on the code and hardware analysis, actions for parallelization and optimization of the NAPOM are proposed.

- In the first stage, the program code needs some corrections. There are some obvious flaws, e.g. accessing the non-existing elements of array. Fortunately, those accesses did not produce fatal errors as the further processing did not take in account values gathered from invalid addresses.
- There are unnecessary operations, e.g. $\left(\sqrt{x^2+y^2}\right)^4$ or $x = y + a \ldots - a, \ldots$ Such operations produce computational overheads as well as additional rounding errors.
- There are multiple similar independent programming loops which could be implemented as a single loop.

- There are unnecessary initializations of large matrix variables as well as intermittent large matrix variables for storing intermediate results that could be replaced with scalar variables.
- The main program code requires a substantial amount of memory; consequently most of the memory operations are performed in the main memory instead of in the cache memory. As shown in the memory performance measurements, the main memory performance is a bottleneck of MBS nodes. On the other hand, the MBS CPUs have relatively large caches at disposal that are not fully utilized in algorithms that only use a single core. Each CPU has two 6MB L2 caches, i.e. pair of cores share one L2 cache. Therefore, the most demanding parts of the code should be redesigned in order to minimize the penalties from accessing the main memory. One of the first steps is to use all available caches instead of just half as in current implementation.
- The first step towards fully parallel code is the parallelization of all internal loops, in the sense that the highest possible number of cores that share the same memory will be engaged in the parallel program execution. In this way the computational resources increases with the number of cores, however with small penalties for non-intensive inter-core communication. There are several independent executions of the almost identically complex modules (advt called on T and S, advq called on T and S) which can be executed in parallel, within a multi-thread program that will run on more computing cores, with minimal overheads for threads generation. All bigger independent spatial loops can be also executed on more cores to maximize efficiency.
- Second possibility, based on the domain decomposition and a message passing paradigm is not optimal for the existing MBS architecture. Based on the previously presented results from the evaluation of hardware components, the obtained execution times and the characteristics of data structures used in the simulations, the distributed parallelization of the actual NAPOM code is not likely to be effective. The communication overheads would diminish the benefits obtained from concurrent execution of the program code on more computational nodes. There is simply not enough computational complexity in comparison to the required communication. Besides that, the domain decomposition would require severe interferences in the code. By our opinion, such an approach is not suitable for current task. However, the distributed parallelization would be interesting if the code would be executed on a cluster with more computational nodes and on more computationally demanding cases (finer grids, additional physical phenomena, etc.).
- The pre-processing phase consumes about one third of the whole execution time and should be thoroughly optimized. The pre-processing phase can be, from the execution time point-of-view, severely improved.
- The pre-process utilizes several independent loops that should be parallelized with the same principles as the main code.

- In the pre-process several independent actions are performed (downloading, decompressing, data processing, etc), which can be executed concurrently to maximize the efficiency of a computing node. The concurrent execution would considerably reduce the execution time of the pre-process scripts.
- The data downloading and processing from SEA server could be divided in two parts and executed on two additional nodes.
- Finally, the compiler is not optimally configured; the compiler flags should be set to speedup the program execution.

With the above proposed actions the NAPOM package should execute much faster and, to some point, more accurate, since unnecessary rounding will be removed. The proposed optimization/parallelization strategy will minimally affect the MBS cluster CPU workload. One node will be, however, fully occupied most of the execution time and only two additional computational nodes will be occupied during the pre-process phase. Considering the execution time of the original sequential code, the resulting parallel code will effectively lower the power consumption of the MSB cluster, while computing the same amount of results.

4 Implementation of proposed improvements

4.1. Pre-process

After the analysis and planning phase, the implementation of proposed improvements takes place. For the sake of readability only the major actions are presented.

First, the pre-process is redesigned. The considerable amount of effort in the pre-process optimization has been put into synchronization of all modules. The downloading, decompressing and data processing are now concurrent, which considerably reduces the execution time. All the concurrent runs of the scripts have been tuned to maximize the efficiency. Three computational nodes are used in the pre-process phase. Besides the head node, where most of the work is done, the data from SEA is divided in two parts and distributed among additional two computers (host 1 and host 2). The workflow diagram of optimized pre-process phase is schematically presented in Figure 6.

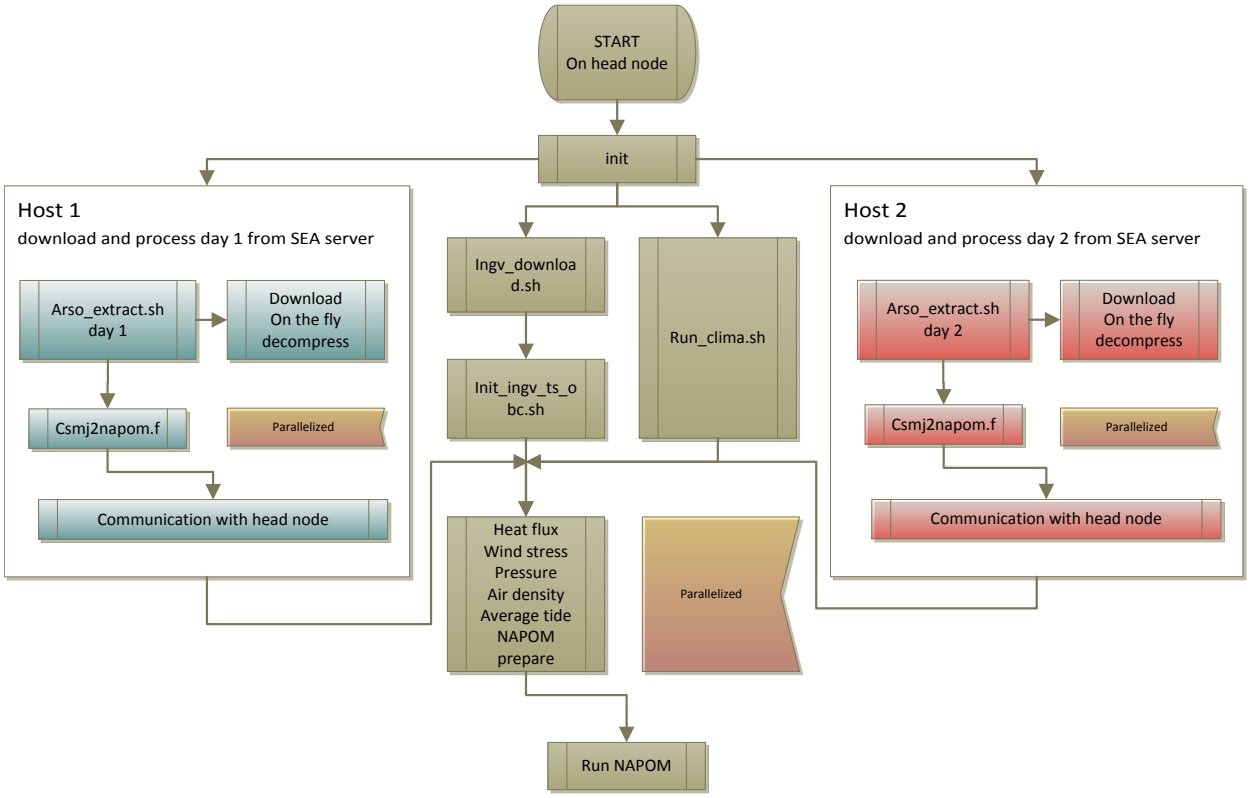

Figure 6: Diagram of optimized pre-process phase

Besides the improvements presented in Figure 6, several other improvements have been implemented, e.g. changing download protocol to achieve better performance, on the fly decompression, removing unnecessary numerical operations, etc.

4.2. Main code

Next, the improvements in the main NAPOM code are shown in Figure 7. Most of the demanding operations with high memory usage have been decomposed over y coordinate in order to maximize the cache hit rate. Independent operations with small memory footprint have been parallelized to run concurrently with OpenMP API. Loops with larger memory footprints that shared a set of variables were combined and converted into the smallest possible number of parallel loops. The sequential code has been redesigned into multi-threaded code, which exploits full capabilities of a single computer. All the cores are involved in computations with an optimal reordering of computation regarding the cache architecture. The diagram presented in Figure 7 is based on the block diagram from POM manual. Please note that only changed modules are shown..

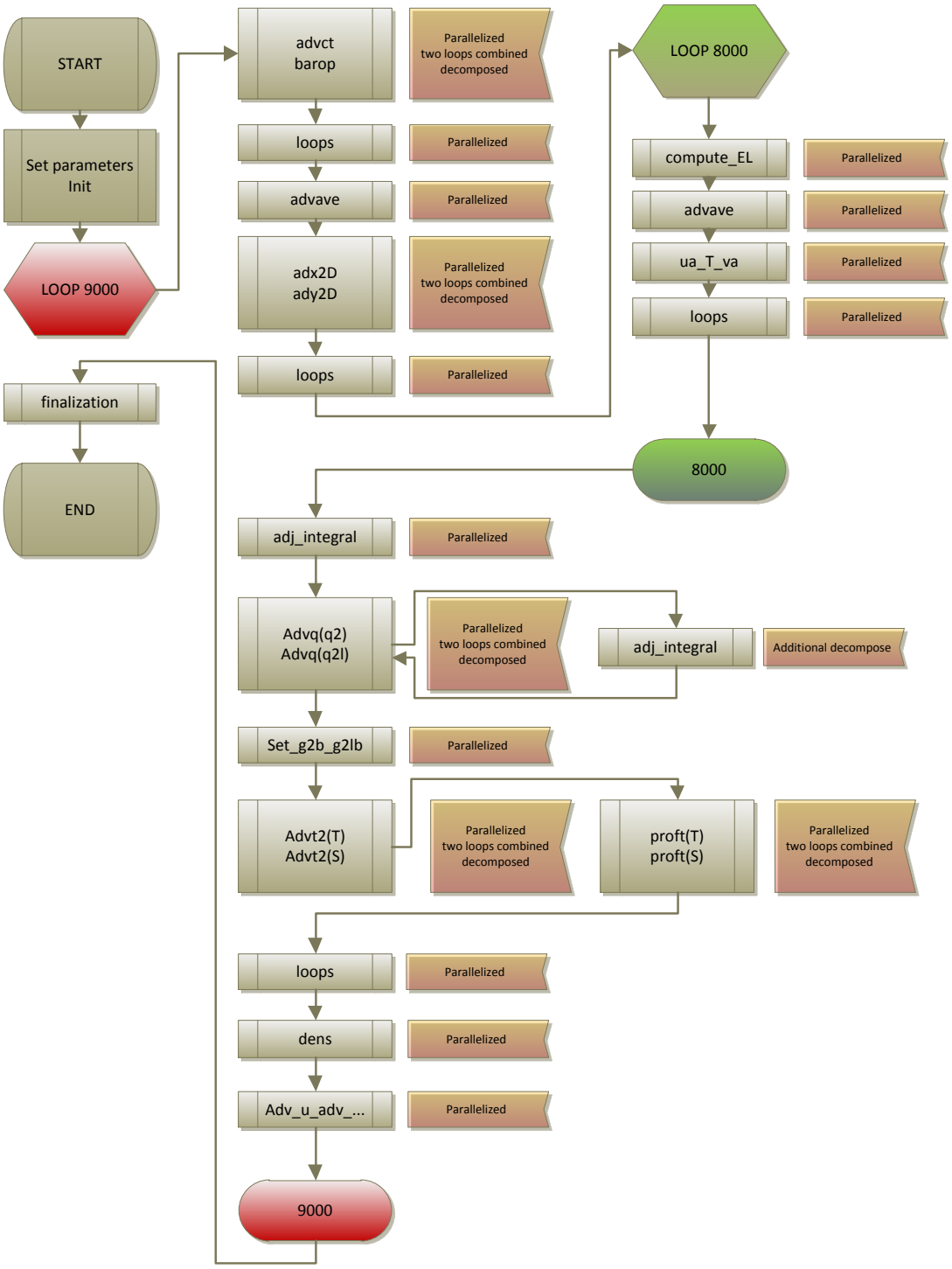

Figure 7: Diagram of optimized NAPOM main code

Besides the presented actions, there are several smaller improvements implemented, e.g removing unnecessary mathematical operations, removing unnecessary temporary matrices, reducing unnecessary memory copying, etc.

Finally, the compiler and run time environment are set up as shown in Table V.

Table V: Compiler and run-time setup.

Compiler:	-Mextend -O4 -fast -mp=numa -Mquad -Mnofpmisalign -Mdse
run time environment:	ulimit -s unlimited export MP_BIND="yes" export MP_BLIST="0,1,2,3"

5 Results

5.1. Results of numerical integration

In section 2.4 the stability of the NAPOM package has been discussed. It has been shown that even small interferences in the compiling procedure, e.g., using alternative compiler and/or setting compiler flag -mp, or in the input data cause measurable differences in the final computed results. During the optimization process the NAPOM code has undergone several changes (see section 3), therefore, before we start with the speedup measurements, integration results of original and optimized code are compared. The results of a full 72 hours NAPOM simulation on 23.11.2012 are used for the analysis of optimized code. In Figure 8 the qualitative comparison of the original and optimized calculations is presented for *w* velocity on layer 4. The differences are minimal. More detailed quantitative comparison is introduced in Figure 9, where six physically relevant fields are analysed (density, salinity, temperature and all three velocity components). Note that the variables are named according to the output file that represents the results of simulation. Figure 9 comprises two plots, the left one stands for the analysis of differences in results, which were observed after the perturbation of a single input file (see section 2.4) and the right one stands for the analysis of differences observed after the complete run of the final version of the fully optimized and parallelized NAPOM package. As expected, the differences are higher in the final version of NAPOM package, which includes multiple changes to the pre-processing and the main NAPOM code that all influence rounding, compiler optimization switches, usage of advanced execution models (Streaming SIMD Extensions (SSE) or Advanced Vector Extensions (AVX)), etc. However, the differences are within the same ranges. A significant effort is needed, which is beyond the scope of this project, to exactly determine which results, original or optimized, are more accurate, as the optimized code performs significantly less rounding. It is also important to interpret the presented comparison in a proper way. The

subtraction of two fields is not necessarily a good measure for the accuracy of results. In the presented case we are dealing with highly non-linear and coupled system, which may result in highly unpredictable output. The results could be different if the same code with the same input would be executed on different computers with different hardware architectures. The goal of the presented comparison is to show that the optimized code provides the simulation results, which are within the same ranges as the original one and that during the optimization process we did not implement eventual bugs or inconsistencies that would corrupt the simulation results. The stability of the NAPOM code is interesting and should be further investigated.

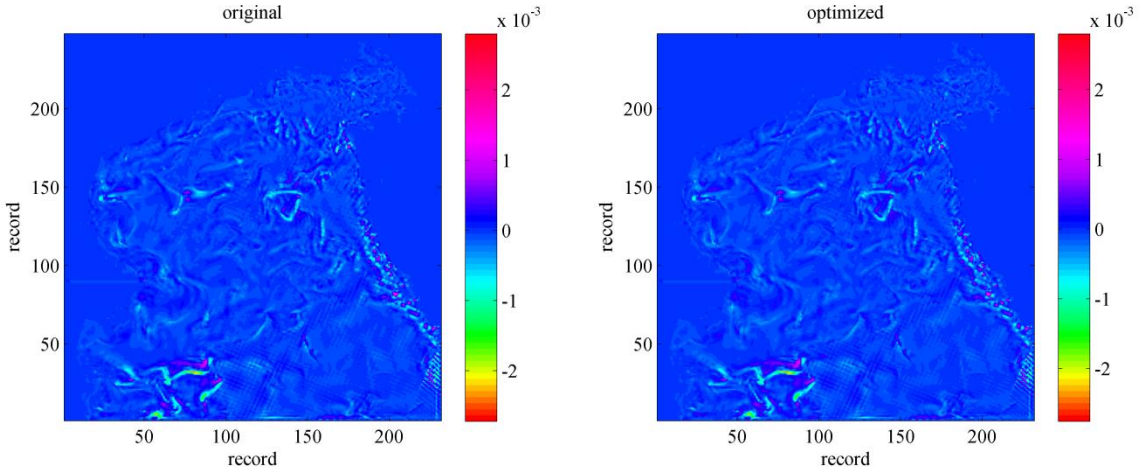

Figure 8: Comparison of results from original and optimized NAPOM code for w velocity on layer 4.

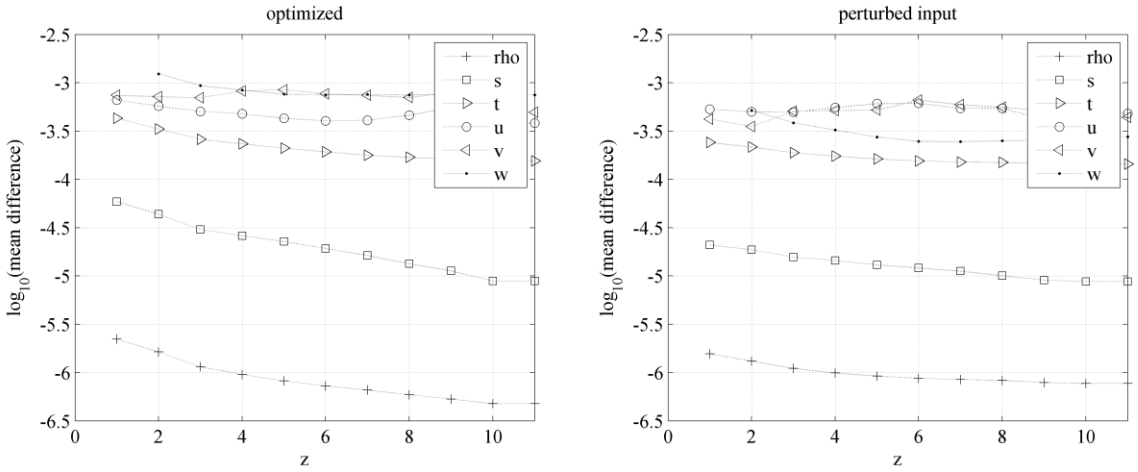

Figure 9: Difference between original calculation with slightly perturbed data (left) and optimized calculation with all implemented optimization steps (right), for different fields.

5.2. Execution performance

The results of implemented optimization process are presented in terms of the execution times of original and optimized code as well as the speedup. In Figure 10 comparison of execution times for main modules is represented. We use the same definitions for the pre-process modules as given in Table I and for the main code as given in Table II. Please note, that the execution times of several modules in the pre-process phase (e.g. first six modules) are reduced to zero as they are executed concurrently.

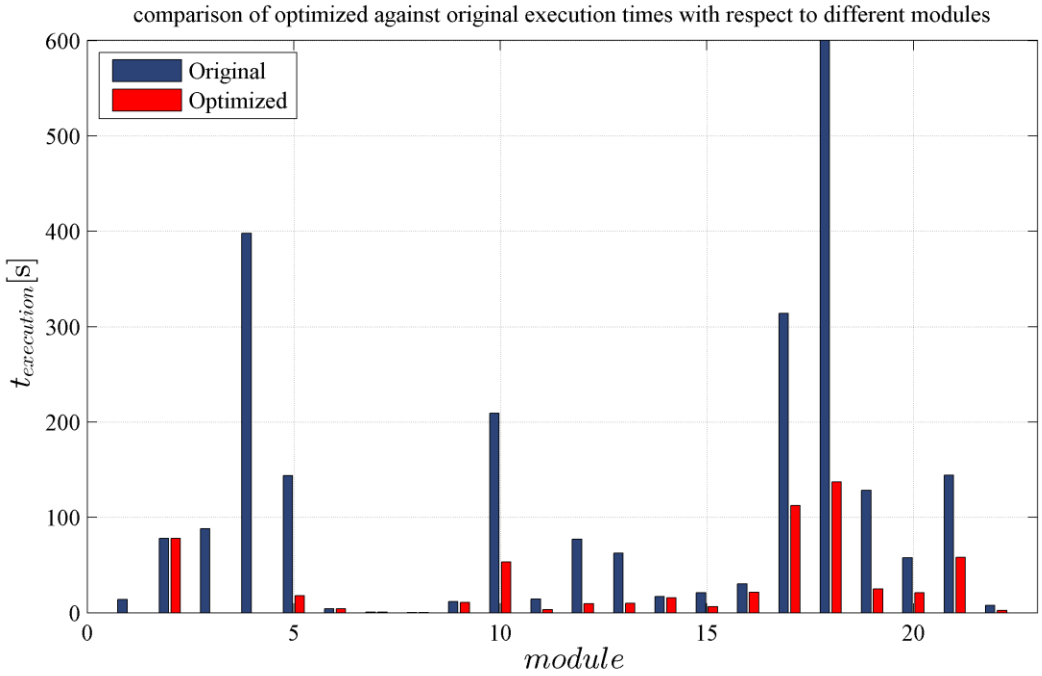

Figure 10: Comparison of original against optimized execution times.

From Figure 10 it is evident that most of the modules have been severely improved. The pre-process phase execution time is reduced from roughly 12 minutes to a bit less than 2 minutes mostly on the account of well-tuned concurrent execution of the completely independent processes (Figure 6). Additional speedup is gained through the shared memory parallelization of local loops. The speedup of the pre-process is about 5.

The execution time of the main code is reduced from the original half an hour run to about 8 minutes run. The speedup of the main code is roughly 3.6. Besides several optimizations (described in Section 3) most beneficial action in the main code is the decomposition and parallelization of huge independent loops, e.g. the computations of lateral viscosity that are reduced from 210 s to 53 s. The exploration of full cache capabilities together with full multicore CPU power severely reduces the execution time. However, appropriate coding actions have to

been implemented for desired effect. Merely adding the OpenMP programming directives will, in most cases, even reduce CPU efficiency. If the processes are not binded appropriately and datasets are not chosen to maximize the multicore efficiency, the thread generation overheads and communication between cores (seen as additional cache misses, especially in L1 cache), might reduce the overall execution efficiency.

To confirm the results, 23 runs with 23 different datasets have been tested. In another words, "operative runs" from 26.9.2012 to 9.11.2012 have been observed. Results are presented in Figure 11, where speedup of a whole NAPOM package regarding different runs together with the execution time is plotted. The speedup is within interval [3.70, 4.06] with an average value of 3.89. In all test runs the difference in computed results, between the original and optimized code, is within the allowed margins (see section 2.4). The small variations in speedup, during the subsequent runs, are due to different factors where the most pronounced is the download time, however, in all tested runs the speedup was higher than 3.5.

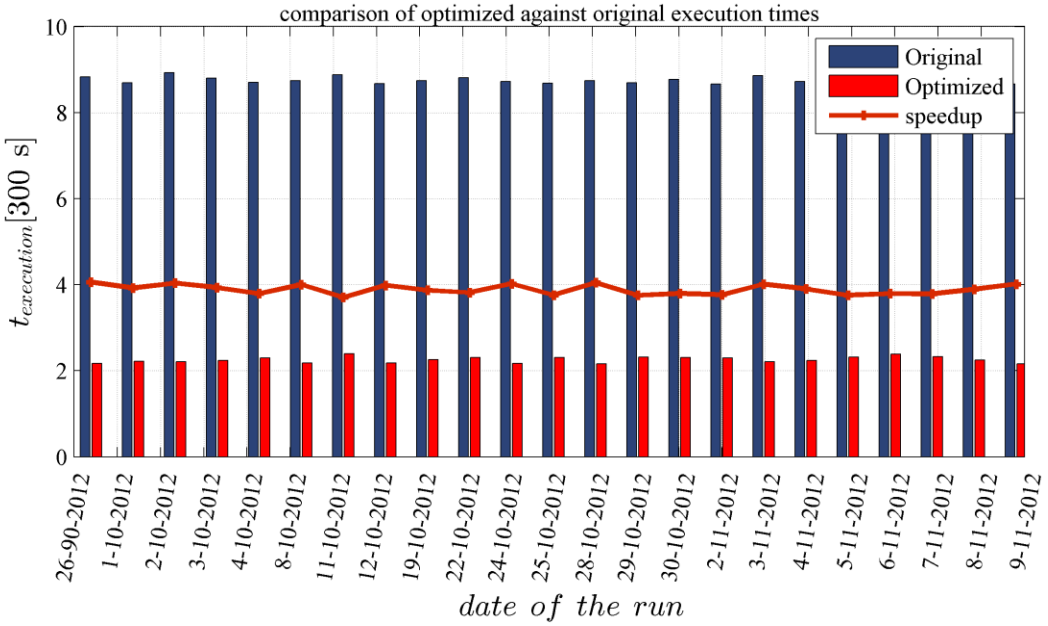

Figure 11: Speedup behaviour of NAPOM regarding different runs

6 Conclusions

Within the project the NAPOM package has been analysed and optimised regarding the MBS computational resources. The execution times of the optimized code are nearly four times shorter

than of supplied code. The introduced errors are within the rounding margins. The optimized package is ready for operative runs on the MBS cluster on the head node, in directory /home/gkosec/num/, organized as

- NAPOM_orig_v1 – first version of the supplied code
- NAPOM_orig_v2.tgz – second version of the supplied code
- NAPOM_orig_v2_patched –second version of teh supplied code with applied patches needed to execute the code (supplied code does not execute). In the file orig_v2_to_orig_v2_pached.diff all the applied changes can be found.
- NAPOM_optimized – optimized package.

The scripts for running NAPOM follow the naming convention of the original scripts, while the main script of the package is named run.all.parallel.sh. The user that wants to run the parallel NAPOM must have the same permissions as for the original version and must be able to use the newly installed PGI Fortran compiler 12.8 (both can be set up by the system administrator).

To keep a reference and to aid future changes to the code, the parallel source code also includes two versions of all functions and subroutines that were parallelized – the original serial version and the new parallel version.

To summarize, the main actions implemented during the project are:

- The MBS cluster and supplied code has been analysed.
- The optimization and parallelization strategy has been formulated regarding the limitations, goals, and available resources.
- The pre-process has been synchronized, distributed over three computing nodes, parallelized and optimized.
- The main code has been parallelized on all cores of a single head node, decomposed and optimized with a minimal additional cluster workload.
- The whole execution time of NAPOM package has been reduced from 42 minutes to 11 minutes. More precisely, the average speedup over 23 runs is 3.89.
- The preliminary analysis of the NAPOM behaviour has been done to set the error tolerance. It has been confirmed that the single precision rounding error considerably affects the numerical integration.
- Some fatal bugs regarding the addressing of non-existing elements in arrays were detected and fixed.

There are several possibilities for further development on the topic:

- The main code should be redesigned. It seems that the most reasonable approach would be the installation of the newer version of POM and the implementation of NAPOM augmentation into the new environment.
- The main code should be written in double precision (that would be automatically solved if a newer version of POM would be applied).
- Regarding the stability issues it seems reasonable to use more computational points, which would introduce more computational complexity, and could justify the parallelization over more computing nodes.

Further development regarding the computational performance depends on the development of the model. If the complexity will be increased (more points) it might be reasonable to implement hybrid OpenMP-MPI parallelization. There are also possibilities to employ Graphics Processing Units (GPUs) through CUDA or OpenCL APIs, but again, with the current complexity one should not expect severe speedups, as the communication overheads between the CPU and GPU will be prevailing.

If MBS decides to upgrade only its existing computational resources we advise a multi CPU server with an improved memory performance. However, the details about eventual upgrades should be discussed regarding the strategy of further development, as well as the available computer technology at that time.

This is the final report that summarizes the code optimization/parallelization of the NAPOM simulation package. The work has been done in Laboratory for Parallel and Distributed Computing - Department E6 at Jožef Stefan Institute.

References

[1] The Princteon Ocean Model. http://www.aos.princeton.edu/WWWPUBLIC/htdocs.pom/.

[2] Őzisik MN. Finite Difference Methods in Heat Transfer. Boca Raton: CRC Press; 2000.

[3] Kamenkovich VM, Nechaev DA. On the time-splitting scheme used in the Princeton Ocean Model. J Comput Phys. 2009;228:2874–905.

[4] PGI. The Portland group. http://www.pgroup.com/index.htm.

[5] Mellor GL. Users guide for a three dimensional, primitive equation, numerical ocean model: Program in Atmospheric and Oceanic Sciences, Princeton University; 1998.