Centro Euro-Mediterraneo sui Cambiamenti Climatici

Research Papers
Issue RP0096
January 2011

Scientific Computing and Operation (SCO)

NEMO-MED: OPTIMIZATION AND IMPROVEMENT OF SCALABILITY

By **Italo Epicoco**
University of Salento, Italy
italo.epicoco@unisalento.it

Silvia Mocavero
CMCC
silvia.mocavero@cmcc.it

and **Giovanni Aloisio**
CMCC
University of Salento, Italy
giovanni.aloisio@unisalento.it

This work has been funded by the EU FP7 IS-ENES project (project number: 228203) and partially by HPC-EUROPA2 project (project number: 228398) with the support of the European Commission Capacities Area - Research Infrastructures Initiative.The authors thankfully acknowledge the computer resources, technical expertise and assistance provided by the Barcelona Supercomputing Center, namely prof. Jose Maria Baldasano, prof. Jesus Labarta and their stuff members. The research leading to these results has received funding from the Italian Ministry of Education, University and Research and the Italian Ministry of Environment, Land and Sea under the GEMINA project.

SUMMARY The NEMO oceanic model is widely used among the climate community. It is used with different configurations in more than 50 research projects for both long and short-term simulations. Computational requirements of the model and its implementation limit the exploitation of the emerging computational infrastructure at peta and exascale. A deep revision and analysis of the model and its implementation were needed. The paper describes the performance evaluation of the model (v3.2), based on MPI parallelization, on the MareNostrum platform at the Barcelona Supercomputing Centre. The analysis of the scalability has been carried out taking into account different factors, such as the I/O system available on the platform, the domain decomposition of the model and the level of the parallelism. The analysis highlighted different bottlenecks due to the communication overhead. The code has been optimized reducing the communication weight within some frequently called functions and the parallelization has been improved introducing a second level of parallelism based on the OpenMP shared memory paradigm.

Keywords: Oceanic climate model; Profiling; Optimization; MPI; OpenMP

JEL: C63

INTRODUCTION

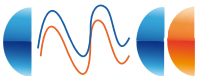
NEMO (Nucleus for European Modeling of the Ocean) is a 3-dimensional ocean model used for oceanography, climate modeling as well as operational ocean forecasting. It includes other sub-component models describing sea-ice and biogeochemistry. Many processes are parameterized, e.g. convection and turbulence. It is used by hundreds of institutes all over the world. The open-source code consists of 100k lines of code, it is developed in France and UK by the NEMO development team and it is fully written in Fortran 90. The MPI (Message-Passing Interface) paradigm is used to parallelize the code. NEMO [5] is a finite-difference model with a regular domain decomposition and a tripolar grid to prevent singularities. It calculates the incompressible Navier-Stokes equations on a rotating sphere. The prognostic variables are the three-dimensional velocity, temperature and salinity and the surface height. To further simplify the equations it uses the Boussinesq and hydrostatic approximations, which e.g. remove convection. It can use a linear or nonlinear equation of state. The top of the ocean is implemented as a free surface, which requires the solution of an elliptic equation. For this purpose, it uses either a successive overrelaxation or a preconditioned conjugate gradient method. Both methods require the calculation of global variables, which incurs a lot of communications (both global and with its nearest neighbors) when multiple processors are used. The scientific literature reports several performance analyses of NEMO model, using different configurations and with several spatial and time resolutions. At the National Supercomputer Centre (NSC) the porting, optimization, tuning, scalability test and profiling of NEMO model on linux - X86-64 - infiniband clusters, have been performed. ORCA1 configuration available from NOCS website has been used for scaling/benchmark studies. Within the PRACE [1] project, a benchmark activity report on several applications has been produced. The NEMO code has been ported and evaluated on several architectures such as the IBM Power6 at SARA, the CRAY-XT4, the IBM BlueGene [6]. The paper describes the research we have carried out on NEMO model with the following goals: (i) the analysis of the code with respect to the MareNostrum target machine; (ii) the optimization of some functions increasing communication overhead; (iii) the improvement of the parallel algorithm in order to better exploit a scalar massive parallel architecture. The paper is organized as follows: the next session describes the NEMO configuration we have used as reference and its performance profiling on the MareNostrum platform; the optimization of the code is detailed in the further section. The last section of the paper describes the introduction of a second level of parallelism and derived benefits.

ANALYSIS OF SCALABILITY

The analysis of scalability of the parallel code aims at verifying how much it is possible to increase the complexity of the problem, in terms of spatial and time resolution, scaling up the number of processes. As first step of the analysis we profiled the original NEMO code for highlighting possible bottlenecks slowing down the efficiency, when the number of processes increases.

MODEL CONFIGURATION

The NEMO configuration taken into account is based on the official release (v3.2) with some relevant improvements introduced by INGV (Istituto Nazionale di Geofisica e Vulcanologia - Italy). Moreover it is tailored on the Mediterranean Basin. The Mediterranean Sea is both too complex and too small to be adequately



resolved in global-scale climate and ocean-only models. To properly address some key processes, it is necessary to adequately represent the general circulation of the Mediterranean basin, the fine-scale processes that control it (e.g. eddies and deep convection), and the highly variable atmospheric forcing. A high-resolution general circulation model of the Mediterranean Sea has been developed in the last 10 years to provide operational forecast of the ocean state [8]: the Mediterranean ocean Forecasting System [4]. The physical model is currently based on version v3.2 of NEMO and is configured on a regular grid over the Mediterranean basin plus a closed Atlantic Box. Horizontal resolution is 1/16 x 1/16 degrees with 72 vertical Z-levels. The model is forced with meteorological data that are either read from gridded external datasets or interpolated on line to the model grid. The model salinity and temperature fields along the boundary of the Atlantic box are relaxed at all depth to external data (open boundary conditions [7]). This is done within an area which has an extension of 2¡ at the west and south boundary and 3¡ at the northern boundary (in order to cover all the area of the Gulf of Biscay). This configuration, even more if coupled with biochemical models, poses several computational challenges:

- High spatial resolution with many grid points and a small numerical time step
- Presence of open boundaries, which implies that additional data need to be read by selected sub-domains
- Computation of diagnostic output across sub-domains
- Storage of large amount of data with various time frequencies.

PROFILING

The research activity has been carried out at the Barcelona Supercomputing Center using the MareNostrum cluster. It is one of the most powerful systems within the HPC-Ecosystems in Europe. It has a calculation capacity of 94.21 Teraflops. One of the key issues that characterize MareNostrum is its orientation to be a general purpose HPC system. The computing racks have a total of 10240 processors. Each computing node has 2 processors PowerPC 970MP dual core at 2.3 GHz, 8 GB of shared memory and a local SAS disk of 36 GB. Each node has a network card Myrinet type M3S-PCIXD-2-I for its connection to the high-speed interconnection and the two connections to the network Gigabit. The default compilers installed are IBM XL C/C++, and IBM XL FORTRAN. There are also available the GNU C and FORTRAN compilers. The MareNostrum uses GPFS as high-performance shared-disk file system that can provide fast, reliable data access from all nodes of the cluster to a global file system. Moreover, every node has a local hard drive that can be used as a local scratch space to store temporary files during the execution of user's jobs. All data stored in these local hard drives will not be available from the login nodes. The first evaluation was focused on establishing how much the computational performance are influenced using the GPFS file system or the local disks. The results, showed in figure 1 and analytically reported in table 1, highlight that the exploitation of local disks can reduce the wall clock time up to 40% against using the GPFS file system. The performances of the GPFS are strictly related to the actual load of the whole cluster and hence they are very variable during the time. Since the local disks are not accessible from the login node, some modifications to the NEMO runscript file, used to launch the model, have been performed.

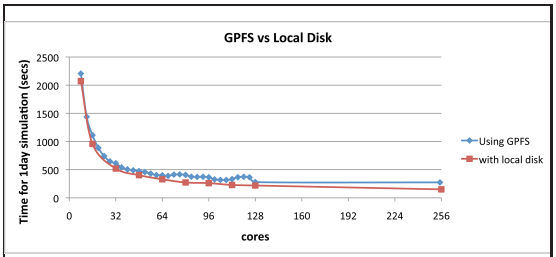


Figure 1:
Execution time on 1day simulation: GPFS vs local disk.

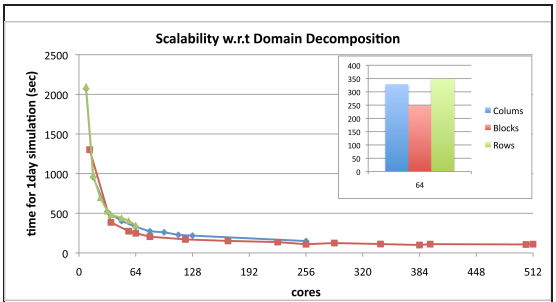


Figure 2:
Execution time on 1day simulation w.r.t. domain decomposition.

Table 1
Execution time on 1day simulation: GPFS vs local disk.

Cores	GPFS	Local Disk	Gain(%)	Speed-up
8	2205.86	2071.45	6.09	1.065
16	1112.19	957.17	13.94	1.162
32	614.02	519.91	15.33	1.181
48	474.65	402.41	15.22	1.180
64	401.29	328.58	18.12	1.221
80	*407.67*	*272.55*	*33.14*	*1.496*
96	365.64	260.42	28.78	1.404
112	331.35	227.85	31.24	1.454
128	279.75	218.35	21.95	1.281

The NEMO code supports 2D domain decomposition. The size and the shape of the sub domain assigned to each parallel process impact on the overall performance. The second step of performance analysis has been focused on the impact of the domain decomposition on the wall clock time. The analysis of the scalability has been performed taking into account a 1D decomposition (both horizontal and vertical) and a 2D decomposition. The 2D decomposition has been chosen such that the local sub domain would have a square shape. The experimental results demonstrate that the best performance is achieved using a 2D decomposition as showed in figure 2.

The overall evaluation of the legacy code has been carried out in order to analyze the parallel behavior of the application. In particular at high level we have taken into account two metrics: the parallel scalability and the parallel efficiency. Both metrics provide an overall evaluation on how much the code is well parallelized. These measures can guide further analysis focused on specific aspects of the code. It is worth noting here that the approach followed for the analysis started considering the application as a black box. The complexity of the code makes quite unfeasible the definition of a reliable theoretical performance model. The approach we followed was based on an experimental approach. The parallel efficiency and speed-up, respectively reported in table 2 and figure 3, have been evaluated taking as reference time the wall clock time of the application with 12 processes. Due to the amount (8 GB) of main memory per node available on MareNostrum, the execution of the sequential version of the model is prohibitive requiring at least 20 GB with this configuration. The experimental results showed a limit of the scalability up to about 192 cores.

In order to perform a deeper investigation on the motivation for the poor efficiency, we have analyzed the scalability of each routine in the code. For identifying those routines with a relevant computational time we used the gprof utility and the Paraver [3] tool with dynamic instrumentation of the code. Figure 4 shows a paraver

Table 2
Original code performance.

Decomposition	Cores	1day sim (sec)	Efficiency (%)	SYPD	1Y sim (hours)
6x2	12	1302.54	100.00	0.18	132.06
16x4	64	248.71	98.20	0.95	25.22
20x6	120	170.37	76.46	1.39	17.27
128x2	256	109.57	55.72	2.16	11.11
32x9	288	124.84	43.47	1.90	12.66
34x10	340	112.28	40.94	2.11	11.38
128x3	*384*	*99.87*	*40.76*	*2.37*	*10.13*
36x11	396	111.08	35.53	2.13	11.26
128x4	512	110.57	27.61	2.14	11.21

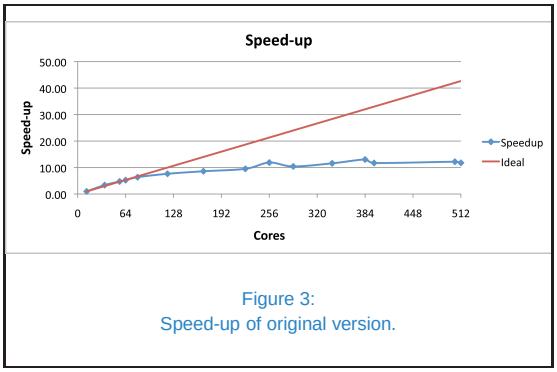


Figure 3:
Speed-up of original version.

snapshot of a very short run (just 3 time steps). Different colors represent different states of the run: blue for computation, red for I/O operations, orange and pink respectively for global and point-to-point communications. *opa_init* initializes the parallel environment and synchronizes processes. Its execution time is negligible when the number of steps is high. The first and the last time steps perform some IO operation, respectively reading input and restart and writing output and restart.

The analysis has been restricted to a single time step: we chose a "general" time step, considering as "general" those time steps with operations occurring every time. Indeed some "occasional" operations like reading the open boundaries values, or storing the state variable values, occur only for some particular time step. We have identified about 36 routine of interest

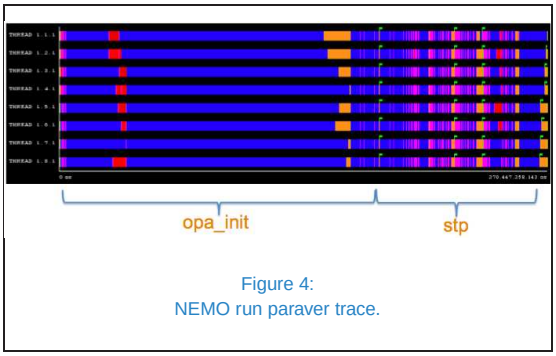


Figure 4:
NEMO run paraver trace.

and we have evaluated their scalability running the application with 8, 16, 36, 72 and 128 processes. For each routine we have taken into consideration both computing and communication time. The results of the analysis are reported in figure 5.

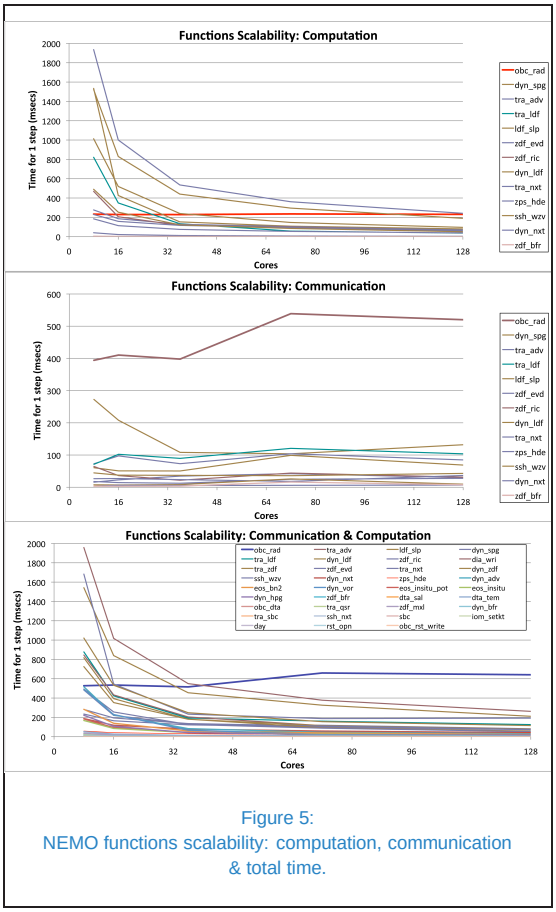


Figure 5:
NEMO functions scalability: computation, communication & total time.

The analysis immediately highlighted the *obc_rad* routine (in charge of calculate the radiative velocity on the open boundaries), the *dyn_spg* (in charge to solve the elliptic equation for the barotropic function) and the *tra_adv* (in charge to evaluate the advection transport of the fields), as those routines to be deeper investigated.

OPTIMIZATION

The optimization phase aimed at redesign critical part of the NEMO code taking into account the following main aspects:

- Exploitation of the memory hierarchy. A relevant limitation of the performance is strictly related to redundant memory accesses or to a high level of cache miss ratio
- The I/O operations are one of the critical factors that limit the performance and the scalability of a climate model. The I/O pattern implemented in NEMO can be classified as: read once and write periodically
- The communication among parallel processes plays a crucial role on the performance of a parallel application. Several good practices can be followed in order to reduce the communication overhead, such as modifying the communication pattern in order to overlap communication and computation; joining several short messages sent with several MPI calls in a bigger one sent once.

The analysis of the scalability showed a limit at 192 cores due to a high level of communication overhead. The bottleneck has been identified in the function responsible for the evaluation of the open boundaries conditions. After the evaluation of the open boundaries, the processes exchange the overlapped values over the boundaries with their neighbors. The function toke more than 60% of its time in communication.

OBC_RAD FUNCTION

As already stated before, the NEMO configuration we used for our analysis is limited to an oceanic region and namely the Mediterranean basin, which communicates with the rest of the global ocean through "open boundaries". An open boundary is a computational border where the aim of the calculations is to allow the perturbations generated inside the computational domain to leave it without deterioration of the inner model solution. However, an open boundary has also to let information from the outer ocean enter the model and should support inflow and outflow conditions. The open boundary package OBC is the first open boundary option developed in NEMO. It allows the user to:

- Tell the model that a boundary is "open" and not closed by a wall, for example by modifying the calculation of the divergence of velocity there
- Impose values of tracers and velocities at that boundary (values which may be taken from a climatology): this is the "fixed OBC" option
- Calculate boundary values by a sophisticated algorithm combining radiation and relaxation ("radiative OBC" option).

The Open Boundaries calculation is performed within the *obc_rad* routine. The current implementation of the *obc_rad* function swaps arrays to calculate radiative phase speeds at the open boundaries and calculates those phase speeds

if the open boundaries are not fixed. In case of fixed open boundaries the procedure does nothing. In particular the following algorithmic steps are performed: (i) each MPI process calculates the radiative velocities on its sub-domain starting with zonal velocity field; (ii) the data on the border of the local sub-domain are exchanged among MPI processes with a cross communication pattern; (iii) repeat from step one for the following fields: tangential velocity, temperature and salinity. In the worst case, when the whole domain has 4 open boundaries (east, west, north and south) each MPI process performs 16 exchanges (4 fields exchanges multiplied by 4 open boundaries). For each field, an MPI process sends and receives the data to/from 4 neighbors. Even though the exchanged fields are 3D arrays, the current implementation of the communication routine (named *mppobc*) calls iteratively a library routine for sending/receiving 2D arrays. Figure 6 shows the original communication pattern among processes.

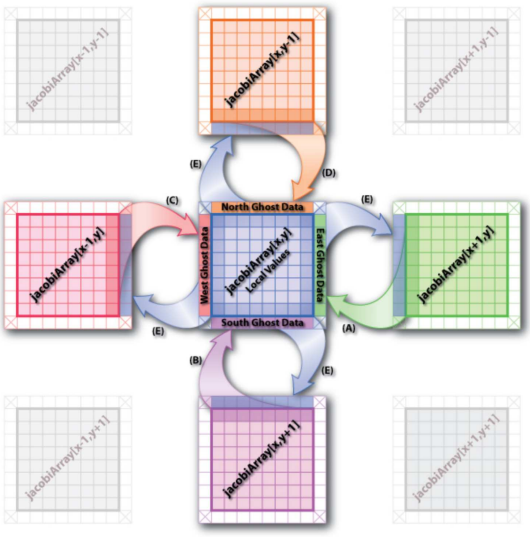


Figure 6:
Communication pattern during the open boundaries evaluation: before optimization all processes were involved in the communication.

OBC_RAD OPTIMIZATION

The analysis of the scalability showed that the communication overhead within the *obc_rad* function reaches a ratio of 74% running the model with 8 cores. The main limits to the scalability have been then identified in a heavy use of communication among processes. With a deeper analysis of the *obc_rad* algorithm we noticed that several calls to the MPI_send/MPI_recv were redundant and hence they could have been removed. Figure 7 illustrates the essential communications needed for exchanging the useful data on the boundaries.

The optimization reduced the communication time through the following actions: the processes on the borders are the only processes involved in the communication; the data exchanged between neighbors are only the data on the boundary; the data along the vertical levels are "packed" and sent with only one communication invocation. Figures 8 and 9 show respectively the how communications (yellow lines) within the *obc_rad* are drastically reduced after the optimization.

The analysis of the scalability of NEMO using the optimized version of the *obc_rad* routine has been performed starting from a configuration on 12 cores with a decomposition 6x2 up to 512 cores (128x4) on 1-day simulation. As reported in table 3, the minimum wallclock time happens on 396 cores with a decomposition 36x11. Efficiency increases compared with the original version, as well as the parallel speed-up (figure 10), and the *obc_rad* execution time was reduced of about 33.81%.

SOL_SOR FUNCTION

After the *obc_rad* optimization, a new detailed analysis of scalability (figure 11) on all of the

Table 3
OBCRAD optimized vs original version: performance analysis.

Decomposition	Cores	Original exec. time (sec)	Original efficiency(%)	OBCRAD optimized exec. time (sec)	OBCRAD optimized efficiency (%)
6x2	12	1302.54	100.00	1281.28	100.00
12x3	36	385.32	112.68	382.47	111.67
14x4	56	274.22	101.79	244.73	112.19
16x4	64	248.71	98.20	226.17	106.22
16x5	80	205.00	95.31	171.54	112.04
20x6	120	170.37	76.46	127.54	100.46
24x7	168	151.45	61.43	95.98	95.35
28x8	224	136.78	51.02	84.95	80.80
128x2	256	109.57	55.72	88.70	67.71
32x9	288	124.84	43.47	87.24	61.20
34x10	340	112.28	40.94	81.26	55.65
36x11	*396*	*111.08*	*35.53*	*73.53*	*52.80*
128x4	512	110.57	27.61	75.02	40.03

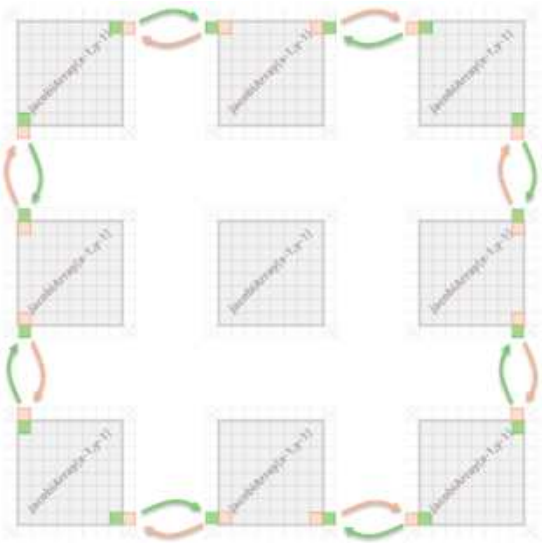


Figure 7:
Communication pattern during the open boundaries evaluation: after optimization only the processes on the boundaries were involved in communication and they exchange only the data on the boundary.

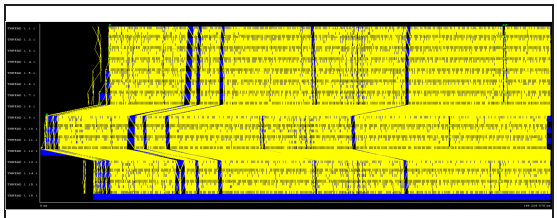


Figure 8:
Communications within OBCRAD before optimization.

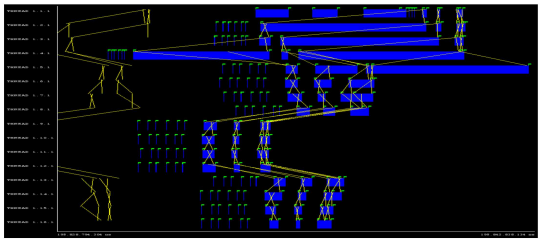


Figure 9:
Communications within OBCRAD after optimization.

above mentioned 36 functions has been performed.

It allowed identifying the SOR solver routine (called by the *dyn_spg* function) as the most expensive from the communication point of view. The function implements the Red-Black Successive-Over-Relaxation method [9], an iterative search algorithm used for solving the elliptical equation for the barotropic stream function. The algorithm iterates until convergence for a maximum number of times. The high frequency of exchanging data within this function increases the total number of communications.

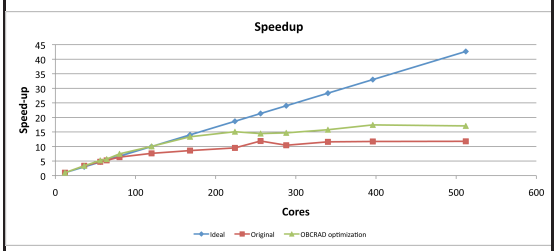


Figure 10:
Speed-up: OBCRAD optimized vs original version.

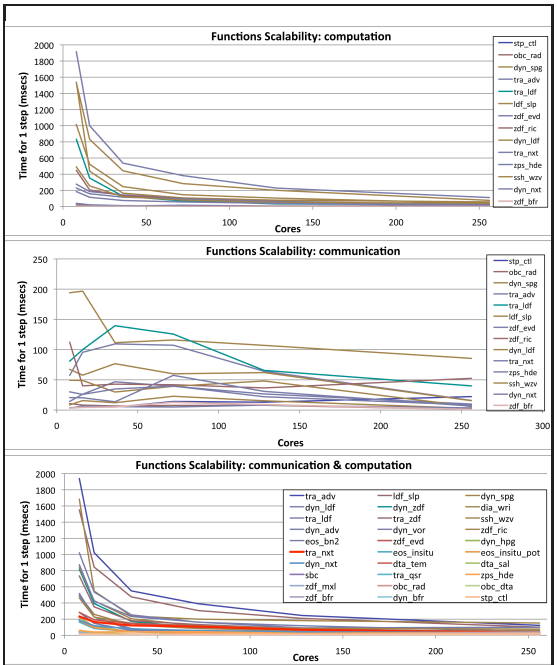


Figure 11:
NEMO functions scalability: computation, communication & total time.

At each iteration, the generic process computes the black points inside the area, updates the black points on the local boundaries exchanging values with neighbors, computes red points inside and finally updates red points on the local boundaries (always exchanging with neighbors). Each process exchanges data with 4 (at north, south, east and west) of its 8 neighbors: the order of data transfer guarantees data reliability. Communications are very frequent and the total number of exchanges is given by the number of iteration multiplied by 2 (one for red points, and one for black) by 4 neighbors. The *sol_sor* function, implementing the SOR solver method, calls the *lnk_2d_e* function for exchanging data among processes. Both the functions are characterized by two components: a running component respectively computing and buffering data before sending and after receiving and a communication one (within the *sol_sor* there is a group communication during the convergence test).

SOL_SOR OPTIMIZATION

The algorithm of *sol_sor* suggests a possibility to improve performance, especially when the number of processes increases. At each iteration, communication and computation could be overlapped. The algorithm can be modified as follows: (i) computing of data on the local boundaries, (ii) communication of computed region overlapped with computation of the inner domain. This solution has been implemented, but it did not give expected results. The original version uses blocking communications during the exchange. The modified version was implemented by the use of non-blocking communications to allow message transfer to be overlapped with computation. As result, not only running, but also communication time was increased after the code modification. Changing communication algorithm, the computation within *sol_sor* has been split in two steps. This generated an access to non-contiguous memory locations with a consequent increase of L1 cache misses. More cache misses means more instructions and then more computing time. Moreover, the introduction of non-blocking communication does not guarantee the order of data exchanging among processes, so that a generic process

needs to communicate not only with the north, south, east and west processes, but also with the diagonal ones, doubling the number of communications. Since communication and computation are overlapped, the increase of communications number should not increase the execution time. However, the behavior of communications on MareNostrum has not been the expected one. Using the Dimemas [2] tool, we have theoretically evaluated the new algorithm on an ideal architecture with the nominal values declared for MareNostrum (figure 12). Even though from a theoretical point of view the new algorithm performed better than the old one, the experimental results did not confirm the expectations. One of the possible motivations could be found to the implementation of the non-blocking communication within the installed MPI library. Moreover it is worth noting here that with a high level of parallelism, the *sol_sor* function has a fine computational granularity, so that the execution time feels the effects of several causes not directly related to the application but also to the system or to the tracking tools and it is very difficult to estimate its behavior. In these cases the only thing is to consider the experimental results, which on MareNostrum highlight better performances of the original version.

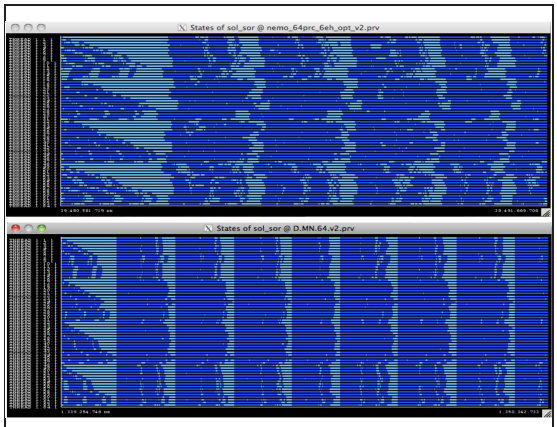


Figure 12:
Analysis by Dimemas: (a) real behavior of communications on MareNostrum, (b) expected behavior of communications on MareNostrum.

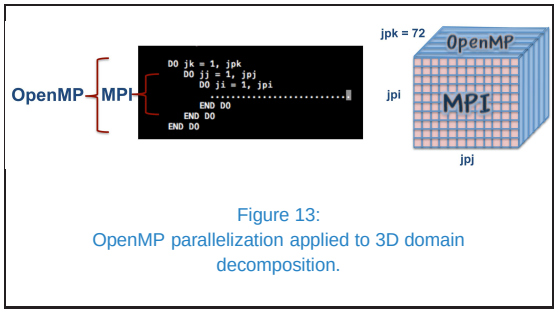


Figure 13:
OpenMP parallelization applied to 3D domain decomposition.

NEMO PARALLELIZATION

Many NEMO routines are characterized by operations performed on a 3D domain, along jpi, jpj and jpk as showed in figure 13. The MPI parallelization exploits the domain decomposition on 2 dimensions (along jpi and jpj). In order to reduce the computational time, a hybrid parallel approach could be introduced. An additional level of parallelization, using the OpenMP shared-memory paradigm, could work on vertical levels, which are fixed for our NEMO configuration to 72.

Before modifying the code, an estimation of the percentage of the application, which should benefit from the use of OpenMP is needed. Using the *gprof* utility the percentage of time spent by functions called by the step routine (simulating a time step) and containing loops on levels without dependences, has been computed. It was about the 83% of the total computational time. OpenMP parallelization has been introduced within all of these functions. Fixing the number of allocated cores, we can execute the application using only MPI (in this case the number of MPI processes will be equal to the allocated cores) or using MPI/OpenMP parallelization (in this case, in order to better exploit the MareNostrum architecture, we created 4 OpenMP thread for each MPI process). A

Paraver analysis of the duration of the functions called in the main loop over the time steps has been performed. Functions execution is more balanced among threads using the hybrid version due to the reduced number of communications (figure 14 shows how time spent waiting for communication, the white lines, has been reduced). The total execution time has been reduced too (figure 15).

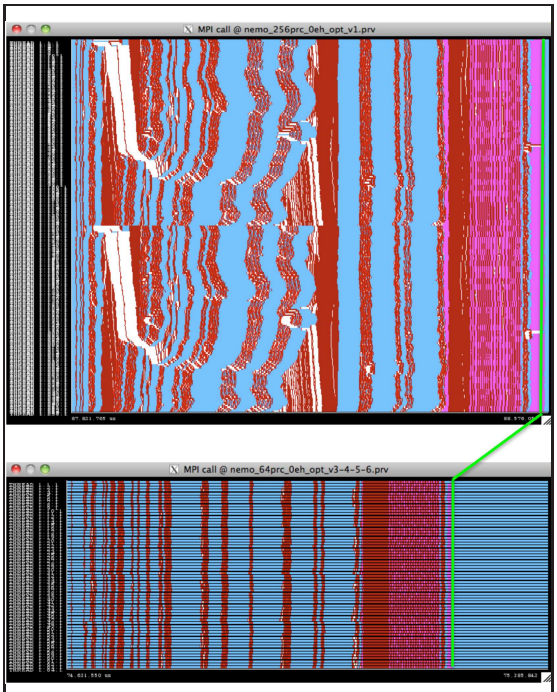


Figure 14:
OpenMP parallelization: (a) communications using 256 cores for 256 MPI procs, (b) communications using 256 cores for 64 MPI procs, each one with 4 threads.

With the OpenMP parallelization, the parallel speed-up improved, as shown in figure 16. The benefits derived from the hybrid parallelization can be appreciated when the number of MPI processes exceeds 30 and consequently the total threads number exceeds 120.

Table 4 shows performance results in terms of execution time and efficiency, on 1-day simulation, comparing the original code with the version after the optimization of the *obc_rad* routine and after the introducing of the second level of parallelism. The minimum wallclock time happens on 396 cores. Efficiency increases compared with both the original version and the *obc_rad* optimized one; execution time is reduced of about 18%.

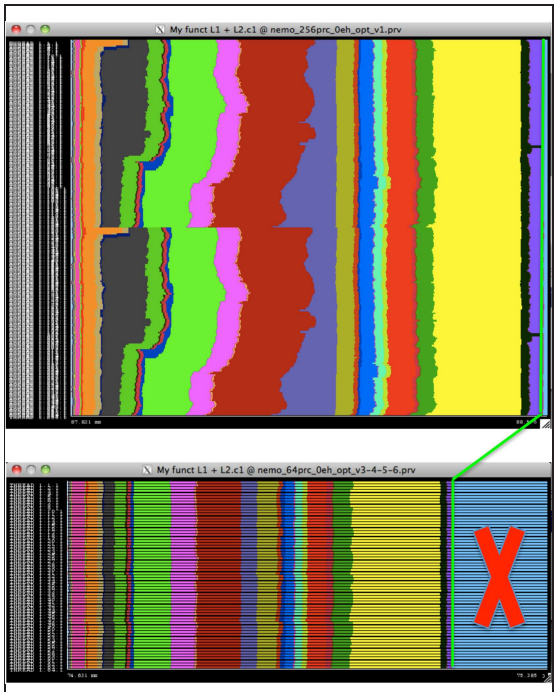


Figure 15:
OpenMP parallelization: functions duration using 256 cores for 256 MPI procs, (b) functions duration using 256 cores for 64 MPI procs, each one with 4 threads.

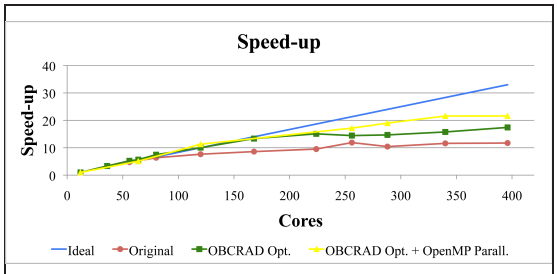


Figure 16:
Speed-up: OpenMP parallelized vs previous versions.

Table 4
OpenMP parallelized vs previous versions: performance analysis.

MPI De-composition	Cores	Original exec. time (sec)	Original ef-ficiency(%)	OBCRAD optimized exec. time (sec)	OBCRAD optimized efficiency (%)	OpenMP exec. time (sec)	OpenMP ef-ficiency (%)
3x1	12	1302.54	100.00	1281.28	100.00	1767.30	100.00
8x2	64	248.71	98.20	226.17	106.22	346.75	95.57
10x3	120	170.37	76.46	127.54	100.46	156.48	112.94
16x4	256	109.57	55.72	88.70	67.71	75.89	80.46
18x4	288	124.84	43.47	87.24	61.20	68.57	79.15
17x5	340	112.28	40.94	81.26	55.65	60.34	76.19
33x3	*396*	*111.08*	*35.53*	*73.53*	*52.80*	*60.29*	*65.47*

Finally, figure 17 highlights execution time for 1-day simulation. Figure zooms on a range of cores between 120 and 396, where all of the optimizations can be more appreciated.

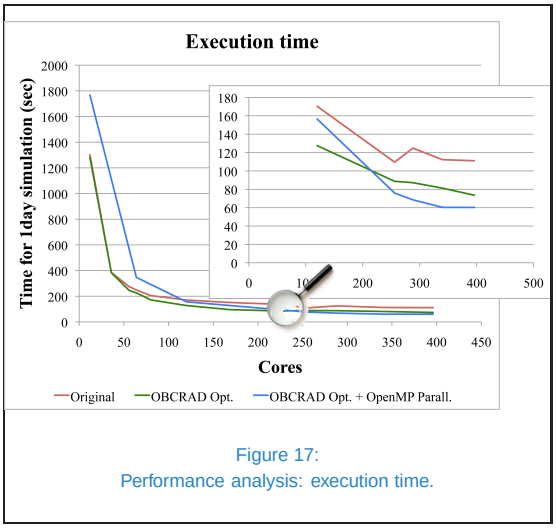
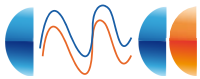


Figure 17:
Performance analysis: execution time.

CONCLUSIONS AND FUTURE WORK

In this work, we presented the profiling and optimization of one of the most deployed oceanic model: NEMO. The profiling phase is mandatory to identify hot-spot functions and to drive optimization. Profiling has been performed on all of the functions called by the main step routine, splitting computation and communication components. This allowed identifying two routines very expensive from the communication point of view: the *obc_rad* and the *sol_sor* routines. The optimization of the *obc_rad* improved execution time of about 34% on 396 MPI processes, while the optimization of the *sol_sor* routine had no benefits on MareNostrum due to the strange behavior of communications on the target machine. The introduction of a second level of parallelism, based on the OpenMP shared-memory paradigm, improved performance of a further 18%. For the future, we plan the following actions:

- The Evaluation of all the optimizations on the IBM Power6 cluster at CMCC
- The evaluation of I/O operations performance, which is a very interesting factor in almost all of the climate codes
- The Integration of all the optimizations within the new version of NEMO-MED based on v3.3 of NEMO
- The evaluation of other levels of optimizations (e.g. at algorithm level).



Bibliography

[1] The partnership for advanced computing in europe.

[2] Barcelona Supercomputing Centre. Dimemas overview. *BSC Performance Tools*, 2010.

[3] Barcelona Supercomputing Centre. Paraver overview. *BSC Performance Tools*, 2010.

[4] INGV. Mediterranean ocean forecasting system.

[5] G. Madec. Nemo ocean engine. Technical Report Technical Report 27 ISSN No 1288-1619, Institut Pierre-Simon Laplace (IPSL), 2008.

[6] P. Michielse, J. Hill, G. Houzeaux, O. Lehto, and W. Lioen. Report on available performance analysis and benchmark tools, representative benchmark. Technical Report PRACE Project Deliverable D6.3.1.

[7] P. Oddo and N. Pinardi. Lateral open boundary conditions for nested limited area models: a process selective approach. *Ocean Modelling*, 2007.

[8] M. Tonani, N. Pinardi, S. Dobricic, I. Pujol, and C. Fratianni. A high-resolution free-surface model of the mediterranean sea. *Ocean Science, Ocean Sci.*, 4:1–14, 2008.

[9] D. M. Young. Iterative methods for solving partial difference equations of elliptic type. *Trans. Amer. Math. Soc.*, 76:92–111, 1954.