IS-ENES3 Milestone M8.4

Definition of NEMO optimization strategy

Reporting period: 01/07/2020 – 31/12/2021

Authors: Italo Epicoco, Silvia Mocavero, Francesca Mele, Mario Acosta, Stella Paronuzzi, Oriol Tintó, Miguel Castrillo, Miroslaw Andrejczuk, Mike Bell

Reviewer(s): Uwe Fladrich, Eric Maisonnave
Release date: 25/01/2021

ABSTRACT

The document describes the work carried out during the first period of the project in terms of performance analysis and optimisation of the NEMO model. In particular, the authors emphasise the need to automatize the performance profiling and define some implementation strategies designed to reduce scalability bottlenecks and the time to solution of the target ocean model. The work is perfectly integrated into the NEMO development strategy plan, which highlights the issues to be addressed in the medium term for the NEMO model, and it contributes to address these issues by defining some optimisations techniques.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824084

Table of contents

1. Objectives

In the context of WP8 which aims at developing models and tools able to produce more accurate and reliable simulations of the Earth climate system, the objective of this task is to improve NEMO computational performance for the execution of high-resolution and complex simulation. This is also in accordance with the NEMO development strategy document [1], where making NEMO efficiently executed on different architectures is defined as one of the main objectives in the five-year period 2018-2022.

This report is aimed at defining a methodology to automatically profile NEMO, in order to understand the main bottlenecks to performance efficiency. Moreover, some solutions to well-known performance issues are suggested and evaluated, also showing the impact that they could have on the NEMO performance. The activities proposed by CMCC and Met Office as internal partners of the NEMO Consortium, alongside BSC as external contributor, aim to analyse the NEMO computational performance and describe (i) a methodology to evaluate which parts of the NEMO code can be safely executed in single-precision and (ii) two complementary strategies to reduce the communication time.

2. Methodology and Results

2.1. Analysis of NEMO computational performance

In order to address NEMO optimisation, which is the main goal of this task, a complete study of the model as numerical and computational code is needed to understand the main bottlenecks of the model and how to appropriately exploit the possible optimizations to be performed during the project. Additionally, NEMO is undergoing significant changes due to the introduction of new developments (such as a completely new ICE module) and some optimizations coming from other projects such as the final implementation (through ESiWACE2[1]) of the mixed precision approach, based on the methodology established in this task (Section 2.3) or tiling implementation (through IMMERSE[2]). For this reason, the main task performed at the Barcelona Supercomputing Center (BSC) during this period has been to study the model and develop methodologies which facilitate work in terms of optimizations and evaluation for the second period of the project (and also in the future), while collaborating on very specific profiling actions, as the scalability of the new BENCH test case or the evaluation of NEMO in the context of the Performance Optimisation and Productivity Centre of Excellence in HPC (POP) Center of Excellence[3]. This section presents a methodology used to automatize the profiling analysis of NEMO to understand the main bottlenecks of the ocean model, not only for a specific version of NEMO, but also

[1] https://www.esiwace.eu
[2] https://immerse-ocean.eu
[3] https://pop-coe.eu

for any version released in the future, including changes coming from other developments and projects. This automatic profiling analysis produces a complete study about the computational performance of the model using the BSC Performance Tools[4], dramatically minimizing the work done by the developers and users.

A methodology to automatically profile the NEMO code

Understanding the cause of parallelization overhead and inefficient computation of a numerical model can be more difficult than it seems. Modern processors and compilers are complex machines that use many strategies in order to do computations as fast as possible, and usually it is not straightforward to see how a change in the code would translate into actual execution. If understanding the performance of a sequential program can be hard enough, things get even more complicated when the program exploits parallelism; and when programs try to exploit supercomputers with thousands of cores instead of just several cores, understanding how computations take place can almost be impossible without the proper tools and methods. At the same time, understanding the main problems of a model from a computational point of view is crucial in order to know how the model uses computational resources and to find ways to improve it. For this reason, the use of profiling analysis tools such as the BSC tools, Vampire or V-Tune is especially critical for these studies.

However, these tools require expertise and their use have some limitations. Almost every year changes in the code or changes in the hardware where scientists run their simulations force us to develop a specific benchmark which could exploit properly the profiling tools to produce useful results. Additionally, it is not clear how to compare the results collected across the years, since no systematic methodology is usually applied.

Having a strategy, agreed upon beforehand, would enable us to consistently profile the performance of a code. We are talking about a methodology which could be used to obtain a deeper insight into the model behaviour while reducing the extra work from experts or to compare the impact of different features in the code on different hardware.

With this purpose, at BSC, we have developed an automatic profiling tool that is able to automatically deploy, compile and analyse a given model, controlled through some input parameters.

This tool aims to be flexible enough to be used with different codes, and on different platforms. It is still in the tuning phase, i.e. understanding what kind of information we need to be able to build a record of BSC codes' performance, and for the moment it can be used just on MareNostrum4 with NEMO. Once the parameter file is filled, mainly with paths to Git repositories and to the script fulfilling the compile step, the analysis is run, and a report is produced.

[4] https://tools.bsc.es/

The report is a latex text file, filled with tables and plots with info as reported in Figure 1. For writing the rest of the report we highly rely on the BSCTOOLS suite, using heavily Paraver or Paramedir to extrapolate data. This methodology will be available for the community along with the optimizations planned for the second part of the project. The users will be able to use this methodology, at least in a simplified version, which should portable enough.

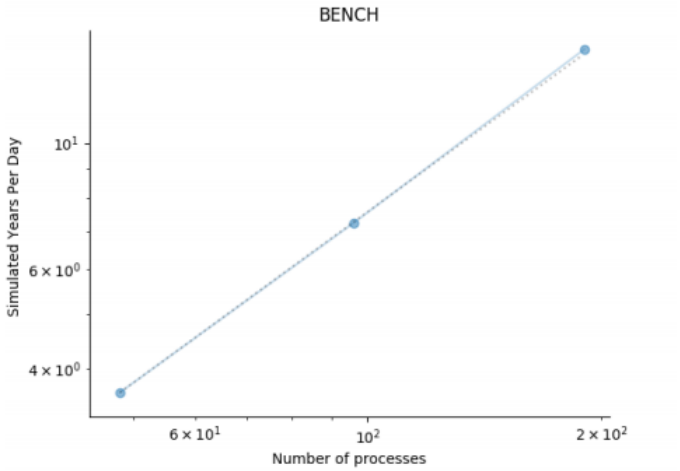

Figure 1: Scalability of nemo model with BENCH

Processes	Comunication	SpeedUp	Efficiency
48	3.6	1.0x	1.00
96	7.2	2.0x	1.00
190	14.6	4.0x	1.02

Table 1: BENCH_scaling

Figure 1 - Report example on BENCH configuration

Apart from the scripts to launch automatically BSC Tools and collect the profiling results, a specific configuration for NEMO was developed. However, we thought that the typical ORCA or GYRE configurations could not be the best ones for a profiling analysis.
During the process of setting up this new automatic profiling methodology, the BSC took part in specific NEMO optimization undertakings, that also allowed us to receive updates on the model necessities and get useful for the methodology.
In 2019, a non-intrusive instrumentation of the NEMO code was created by E. Maisonnave and S. Masson at the LOCEAN Laboratory, in Paris [2], with the aim of giving information about the MPI communications cost and structure of the model. The main goal was to

identify which developments have to be prioritized for the model to enhance its scalability: a new NEMO configuration, called BENCH, was specifically developed for the purpose, offering an easy way to make performance measurements, and hoping to simplify future benchmark activities related to computing efficiency.

We studied if this configuration could be used as a valid tool to get insight into NEMO performance, and then proceeded to use the BENCH test to study some of NEMO most known bottlenecks: I/O and the north fold. Additionally, we took the chance to investigate a topic that is gaining popularity among NEMO developers: the variability of time required to perform a timestep and how it influences NEMO performance, obtaining directions on how to avoid or mitigate such behaviour.

The results state that this test can accurately represent the computing performance of NEMO itself, giving a correct way of interpreting data that may save the experts working on optimization tasks from struggling with the choice of a configuration to assess, or the search for input data as example. We also reported a study that illustrates how completely filling the nodes leads to a loss in performance due to a sort of noise in the time steps duration and explained how this can be related to the need of leaving some space free for the operating system to work.

With respect to the I/O evaluation, we found out that, when running on an ORCA12 configuration using XIOS in server mode, the time steps involved in output operations will slow down with respect to a computation-only one, from six to twelve times. But, it is probably worth exploring additional configurations, because the frequency we used for dumping the output on files was probably too high. Finally, it seems that the north fold extra communications become a burden for the model only at a high number of cores, a region where communications are in general already the bottleneck. In this sense, it is a further remark of the fact that whichever attempt to enhance the parallel performance should necessarily target this topic.

Apart from the preliminary analysis done through the new methodology, in 2020, the NEMO System Team made a request for the POP H2020 Center of Excellence to analyse the main bottlenecks of the NEMO code using also the BENCH test case. The Earth Sciences department of the BSC collaborated to the deployment and configuration of NEMO v4.0.2 in MareNostrum4 for this particular case, also guiding the set-up of the tests and assessing on the scales relevant to the case. This work led to interesting conclusions regarding the communications overhead that can be exploited in this project or in any other attempt to improve NEMO efficiency. In particular, the north fold was again identified as one of the main stoppers to the model scalability, and the final report suggested working on the dynamic solver granularity as a way to reduce waits in the execution.

Final results using the new methodology for profiling analysis will be presented before the end of the project, in order to explain the optimizations that BSC will implement for NEMO

and to provide a complete computational evaluation of the model once the major changes in the forthcoming 4.2 version are implemented.

Performance of the Met Office Global Ocean 8 (GO8) configuration.

Concerning the performance analysis and in particular the I/O operations profiling, at Met Office the GO8 configuration was investigated for a three NEMO ORCA (global grid) resolutions ORCA012 (1/12th degree), ORCA025 (1/4deg) and ORCA1 (1deg), including sea ice but excluding the biogeochemistry module. For all configurations, the impact of the diagnostics load on model execution was investigated. Each configuration was executed one time.

The model configuration was based on NEMO version 4. The runs for ORCA1 and ORCA025 were using 6 XIOS servers, ORCA12 – 36 XIOS servers. All configurations used single file output for diagnostics.

Table 1, Table 2 and Table 3 show results from a series of runs aimed to investigate the impact of diagnostic workload, basic diagnostics in the configuration (Basic) and those required by CMIP6 (CMPI6) on model execution time. Timing information generated by NEMO (timing.output) was used, and this information includes also initialization/finalization time; and XIOS diagnostic file. The execution time was split into the time to run ocean (OCE), Sea Ice (SI3) and XIOS client time (XIOS). Increasing diagnostic workload resulted in an increase of execution time for ocean component by ~15% and increase in XIOS time by 40-60% depending on model resolution. This increase in time is associated with the need to derive/calculate new diagnostics in the model and next to perform temporal averaging and an additional operation in XIOS.

Table 1 - The impact of diagnostic workload on model execution time for ORCA1 configuration

Number of CPUs (XY parallel decomposition)	OCE [s]		SI3 [s]		XIOS [s]	
	Basic	CMIP6	Basic	CMIP6	Basic	CMIP6
2 (2x1)	5222	5993	1197	1193	789	1173
4 (2x2)	2820	322	643	651	407	611
6 (3x2)	2047	2337	455	457	295	435
12 (4x3)	1403	1620	287	284	173	273
28 (6x5)	883	1031	176	174	92	147
34 (6x6)	898	1010	160	158	95	133
48 (9x6)	633	726	109	109	69	103
70 (10x8)	443	505	82	80	56	95
156 (16x12)	196	229	43	42	42	54
322 (24x18)	111	128	24	29	31	49
444 (28x22)	91	107	25	21	34	38

608 (33x26)	81	99	19	22	26	45
754 (36x30)	74	87	17	17	26	35
890 (42x31)	77	83	18	19	32	35

Table 2 - The impact of diagnostic workload on model execution time for ORCA025 configuration

Number of CPUs (XY parallel decomposition)	**OCE [s]**		**SI3 [s]**		**XIOS [s]**	
	Basic	**CMIP6**	**Basic**	**CMIP6**	**Basic**	**CMIP6**
164 (14x15)	6511	7475	1427	1416	602	912
346 (20x24)	2940	3417	576	578	316	494
528 (26x29)	1985	2291	367	368	227	343
702 (30x34)	1549	1778	276	279	186	280
892 (36x37)	1221	1398	217	231	161	246
1072 (37x44)	1034	1174	176	203	149	215
1430 (45x49)	816	968	154	173	124	203
1794 (49x57)	667	776	124	152	115	173

Table 3 - The impact of diagnostic workload on model execution time for ORCA12 configuration

Number of CPUs (XY parallel decomposition)	**OCE [s]**		**SI3 [s]**		**XIOS [s]**	
	Basic	**CMIP6**	**Basic**	**CMIP6**	**Basic**	**CMIP6**
3440 (80x70)	6242	7095	1156	1173	789	1264
4338 (88x81)	5079	5814	889	905	776	1229
6148 (111x93)	3667	4330	617	620	673	1059
7054 (112x106)	3364	3916	587	594	594	962
9748 (134x123)	2554	3112	409	489	530	897

A detailed profiling on time spent running the ocean, sea-ice and I/O component shows (Table 4) that NEMO is the most expensive part of the system – taking between 70% and 76% of total execution time, SI3 takes ~15% and XIOS between 7 and 25% depending on resolution and number of processors used. The data shows that with increasing number of processors relative execution time for NEMO and SI3 is decreasing and for XIOS is increasing.

Table 4 - Relative execution time for OCE, SI3 and XIOS

Resolution	**OCE [%]**	**SI3 [%]**	**XIOS [%]**
ORCA1	72-60	16-14	11-25
ORCA025	76-72	17-13	7-15
ORCA12	76-73	14-12	10-15

2.2. New communication strategies in NEMO

As reported in the previous section and in the NEMO development strategy document, one of the main bottlenecks for NEMO scalability is the time spent performing communications, used to update Lateral Boundaries Conditions. Point to point communications are used by NEMO routines to update the halo region before performing computation on the generic point using values of its neighbours. Two complementary strategies are here proposed by CMCC to reduce the communication frequency and the communication time. In particular, the MPI3 standard defines new neighbourhood collective communications instead of multiple point to point exchanges to perform the halo update. On the other side, the frequency of exchanges can be reduced by increasing the dimension of the halo region.

MPI3 neighbourhood collectives

Lateral boundaries exchange: p2p vs collective communications

NEMO performs Lateral Boundaries Condition (LBC) update by using four point to point MPI communications at north, south, east and west for each MPI domain. NEMO completes east-west exchange before performing north-south communications. The order of the exchanges allows us to preserve both 5-point and 9-point stencils. Indeed, as shown in Figure 2, the bottom right corner of *P0* internal domain (bounded in red) is indirectly received from southern-eastern process *P4* through exchanges performed by the two processes with *P1*.

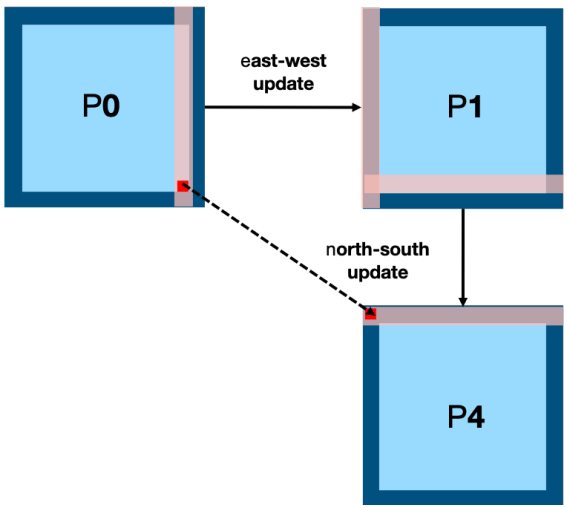

Figure 2 - NEMO LBC update. In the picture we assume to have a 3x3 grid of processes.

NEMO supports the exclusion of computation on land domains through a pre-processing analysis which allows reducing the number of MPI processes allocated at runtime. A new communication strategy should support this feature to preserve performance efficiency.
MPI3 neighbourhood collectives [3] provide a way to have sub-communicators used to perform collective communications.
Two topologies, Cartesian and Graph, are supported and graphically represented in Figure 3. In the case of a Cartesian communicator, a neighbourhood communication involves the nearest neighbours in all directions (north, south, east and west). The neighbourhood communication for a generic process (yellow coloured) on a Cartesian topology includes only the processes that are in green. In the case of Graph topology, the communicator can also involve other processes not directly linked to the target process.

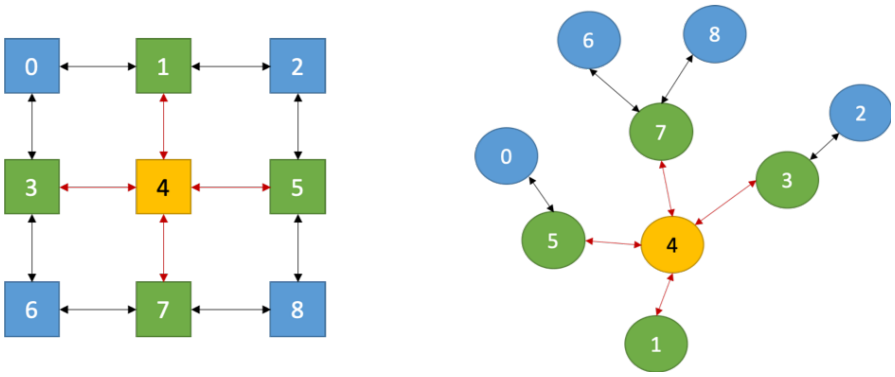

Figure 3 - MPI3 neighbourhood collective communications topologies

Even if NEMO works on a Cartesian grid, Graph topology is recommended to preserve both 5 and 9-point stencils. Neighbourhood communications allow us to support land domain exclusion and communications with eastern and western exchanges when periodicity is not activated by simply excluding the interested processes from the sub-communicator.
The implementation of the new communication strategy only requires changes to the LBC module. Indeed, a parameter in the *lbc_lnk** calls allows the choice between 5 and 9-point stencils, depending on data dependencies in NEMO routines. During the initialization step, two different sub-communicators are defined in order to support the two different exchanges. A single MPI message is needed to be built for all neighbours instead of 4 different messages, as shown in Figure 4, before calling the collective

communication, while the received message is used to update the halo region, following the order of the neighbours in the sub-communicator.

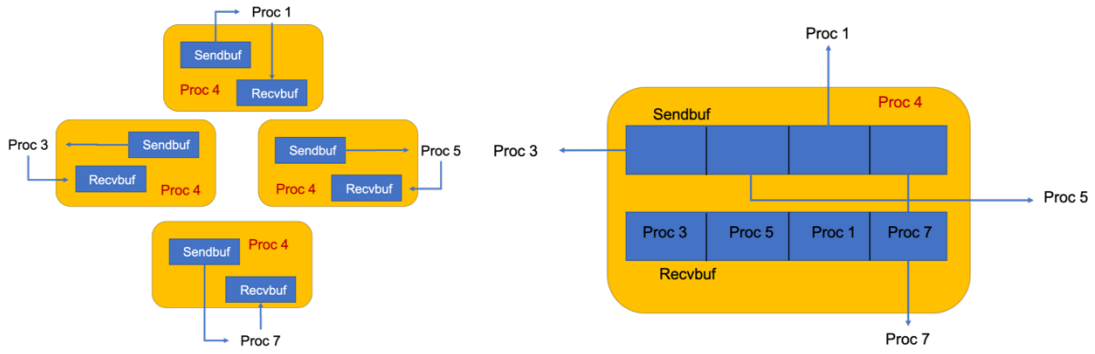

Figure 4 - MPI messages exchanged with current P2P communications (left side) and MPI3 neighbourhood collectives (right side)

Approach evaluation

The new communication strategy has been tested by using a mini-app approach: two computational kernels have been extracted from NEMO and used as test cases. They are the Flux Corrected Transport tracer advection scheme and the computation of ice velocities from EVP rheology, two of the main relevant routines from the computational point of view. The data dependencies are satisfied by the 5-point stencil in the first case while the 9-point stencil is needed to perform computation in the second case.
The new MPI3 neighbourhood collective communications have been integrated into the mini-apps and a performance comparison with the standard MPI2 point-to-point communications has been made. Tests have been performed on a domain size of 3000x2000x31 grid points, by increasing the number of cores up to 2016. This is the limit for the submission queue on Zeus, the CMCC machine where scalability tests have been executed, based on Intel Xeon Gold 6154 18-cores processors at 3.0 GHz. Each computing node includes 2 processors and 96GB of main memory.
Table 5 and Figure 5 show the gain in communication time for the use case 1 using a 5-point stencil. The improvement is not the same when communications with processes on the diagonal are activated (9-point stencil is needed), as shown in Table 6 and Figure 6, however, a modest gain is still achieved. Five repetitions have been executed for each use case and the average response time has been reported. The response time variability is lower than 9%, for both 5 and 9-point stencil.

Table 5 - P2P vs Neighbourhood collectives communication time (5-point stencil)

			MPI3	P2P	
#MPI procs	Horizontal domain size	Vertical domain size	communication time	communication time	Gain (%)
504	107	111	8.67	10.26	15.48
720	83	100	7.50	8.64	13.12
1008	83	72	5.91	7.53	21.51
1440	75	56	5.17	6.76	23.47
2016	54	56	3.55	5.17	31.33

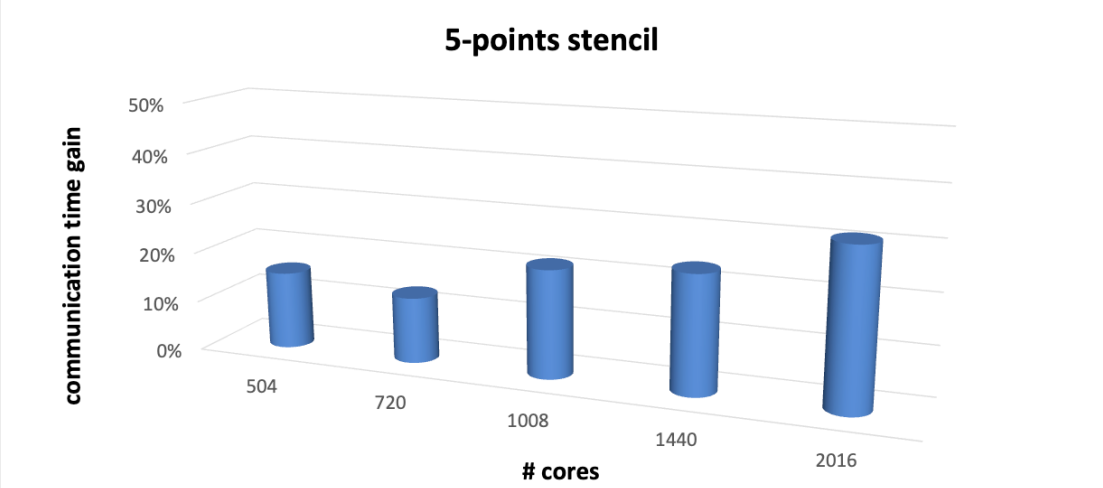

Figure 5 - P2P vs Neighbourhood collectives communication time (5-point stencil)

Table 6 - P2P vs Neighbourhood collectives communication time (9-point stencil)

			MPI3	P2P	
#MPI procs	Horizontal domain size	Vertical domain size	communication time	communication time	Gain (%)
504	107	111	8.25	8.44	2.22
720	83	100	6.88	7.32	5.97
1008	83	72	5.13	6.27	18.22
1440	75	56	4.27	5.04	15.39
2016	54	56	3.70	3.78	1.94

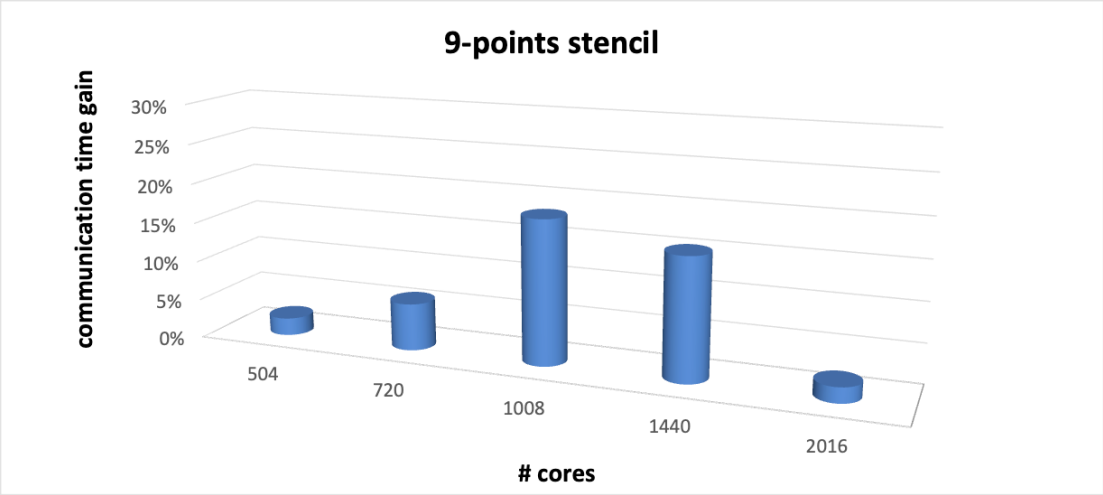

Figure 6 - P2P vs Neighbourhood collectives communication time (9-point stencil)

Extended halo management

Even if the halo size could be parametrised in NEMO, the parallel algorithm in NEMO routines is designed to work with a halo size set to 1 row/column. This means that a new communication is needed whenever the algorithm computes the generic point using its neighbours. The analysis of some NEMO routines shows how the exchange of more than one row/column of halo would allow moving communications outside the routine, preserving data dependencies and reducing communication frequency.
As an example, if we consider the MUSCL (Monotonic Upstream Scheme for Conservative Laws) advection schema, the neighbours access pattern is shown in Figure 7.

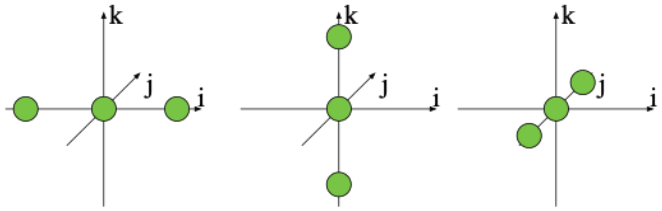

Figure 7 - Neighbours access pattern within the MUSL advection scheme in NEMO

In the target kernel, the horizontal advective fluxes are computed in two steps, each one characterised by the above described access pattern. Then, a communication is needed before each computation region in order to update the halo. Increasing the halo size up to 2 allows us to move communications before the first computation step, then outside the computational kernel, as shown in Figure 8.

```
SUBROUTINE tra_adv_mus( kt, kit000, cdtype, p2dt, pU, pV, pW,            &
      &                 Kbb, Kmm, pt, kjpt, Krhs, ld_msc_ups )

 REAL(wp), DIMENSION(jpi,jpj,jpk,kjpt,jpt), INTENT(inout) :: pt  ! tracers and RHS of tracer equation
 REAL(wp), DIMENSION(A2D(nn_hls),jpk) ::   zwx, zslpx            ! 3D workspace

 DO jn = 1, kjpt            !==  loop over the tracers  ==!
 !
 !                          !* Horizontal advective fluxes
 !
 !                               !-- first guess of the slopes
     zwx(:,:,jpk) = 0._wp                 ! bottom values
     DO_3D( nn_hls, nn_hls-1, nn_hls, nn_hls-1, 1, jpkm1 )
         zwx(ji,jj,jk) = umask(ji,jj,jk) * ( pt(ji+1,jj,jk,jn,Kbb) - pt(ji,jj,jk,jn,Kbb) )
     END_3D

     ! lateral boundary conditions   (changed sign)
     CALL lbc_lnk( 'traadv_mus', zwx, 'U', -1.0_wp )

     zslpx(:,:,jpk) = 0._wp               ! bottom values
     DO_3D( nn_hls-1, 1, nn_hls-1, 1, 1, jpkm1 )    !-- Slopes of tracer
         zslpx(ji,jj,jk) =                    ( zwx(ji,jj,jk) + zwx(ji-1,jj  ,jk) )   &
             &            * ( 0.25 + SIGN( 0.25_wp, zwx(ji,jj,jk) * zwx(ji-1,jj  ,jk) ) )
     END_3D
     !
     DO_3D( nn_hls-1, 1, nn_hls-1, 1, 1, jpkm1 )    !-- Slopes limitation
         zslpx(ji,jj,jk) = SIGN( 1.0_wp, zslpx(ji,jj,jk) ) * MIN(    ABS( zslpx(ji  ,jj,jk) ),   &
             &              2.*ABS( zwx  (ji-1,jj,jk) ), 2.*ABS( zwx  (ji  ,jj,jk) ) )
     END_3D
     DO_3D( nn_hls-1, 0, nn_hls-1, 0, 1, jpkm1 )    !-- MUSCL horizontal advective fluxes
     ! MUSCL fluxes
         z0u = SIGN( 0.5_wp, pU(ji,jj,jk) )
         zalpha = 0.5 - z0u
         zu  = z0u - 0.5 * pU(ji,jj,jk) * p2dt * r1_e1e2u(ji,jj) / e3u(ji,jj,jk,Kmm)
         zzwx = pt(ji+1,jj,jk,jn,Kbb) + xind(ji,jj,jk) * zu * zslpx(ji+1,jj,jk)
         zzwy = pt(ji  ,jj,jk,jn,Kbb) + xind(ji,jj,jk) * zu * zslpx(ji  ,jj,jk)
         zwx(ji,jj,jk) = pU(ji,jj,jk) * ( zalpha * zzwx + (1.-zalpha) * zzwy )
     END_3D

     ! lateral boundary conditions   (changed sign)
     CALL lbc_lnk( 'traadv_mus', zwx, 'U', -1.0_wp )

     DO_3D( 0, 0, 0, 0, 1, jpkm1 )    !-- Tracer advective trend
         pt(ji,jj,jk,jn,Krhs) = pt(ji,jj,jk,jn,Krhs) - ( zwx(ji,jj,jk) - zwx(ji-1,jj,jk  )    &
             &                + zwy(ji,jj,jk) - zwy(ji  ,jj-1,jk  ) ) * r1_e1e2t(ji,jj) / e3t(ji,jj,jk,Kmm)
     END_3D
      ...
 END DO
```

```
SUBROUTINE tra_adv_mus( kt, kit000, cdtype, p2dt, pU, pV, pW,            &
      &                 Kbb, Kmm, pt, kjpt, Krhs, ld_msc_ups )
 !!----------------------------------------------------------------------
 REAL(wp), DIMENSION(jpi,jpj,jpk,kjpt,jpt), INTENT(inout) :: pt  ! tracers and RHS of tracer equation
 REAL(wp), DIMENSION(A2D(nn_hls),jpk) ::   zwx, zslpx            ! 3D workspace
 !!----------------------------------------------------------------------

 CALL lbc_lnk( 'traadv', pt(:,:,:,:,Kbb), 'T', 1.)

 DO jn = 1, kjpt            !==  loop over the tracers  ==!
 !
 !                          !* Horizontal advective fluxes
 !
 !                               !-- first guess of the slopes
     zwx(:,:,jpk) = 0._wp                 ! bottom values
     DO_3D( nn_hls, nn_hls-1, nn_hls, nn_hls-1, 1, jpkm1 )
         zwx(ji,jj,jk) = umask(ji,jj,jk) * ( pt(ji+1,jj,jk,jn,Kbb) - pt(ji,jj,jk,jn,Kbb) )
     END_3D
     !                                !-- Slopes of tracer
     zslpx(:,:,jpk) = 0._wp               ! bottom values
     DO_3D( nn_hls-1, 1, nn_hls-1, 1, 1, jpkm1 )
         zslpx(ji,jj,jk) =                    ( zwx(ji,jj,jk) + zwx(ji-1,jj  ,jk) )   &
             &            * ( 0.25 + SIGN( 0.25_wp, zwx(ji,jj,jk) * zwx(ji-1,jj  ,jk) ) )
     END_3D
     !
     DO_3D( nn_hls-1, 1, nn_hls-1, 1, 1, jpkm1 )    !-- Slopes limitation
         zslpx(ji,jj,jk) = SIGN( 1.0_wp, zslpx(ji,jj,jk) ) * MIN(  ABS( zslpx(ji  ,jj,jk) ),   &
             &              2.*ABS( zwx  (ji-1,jj,jk) ), 2.*ABS( zwx  (ji  ,jj,jk) ) )
     END_3D
     DO_3D( nn_hls-1, 0, nn_hls-1, 0, 1, jpkm1 )    !-- MUSCL horizontal advective fluxes
     ! MUSCL fluxes
         z0u = SIGN( 0.5_wp, pU(ji,jj,jk) )
         zalpha = 0.5 - z0u
         zu  = z0u - 0.5 * pU(ji,jj,jk) * p2dt * r1_e1e2u(ji,jj) / e3u(ji,jj,jk,Kmm)
         zzwx = pt(ji+1,jj,jk,jn,Kbb) + xind(ji,jj,jk) * zu * zslpx(ji+1,jj,jk)
         zzwy = pt(ji  ,jj,jk,jn,Kbb) + xind(ji,jj,jk) * zu * zslpx(ji  ,jj,jk)
         zwx(ji,jj,jk) = pU(ji,jj,jk) * ( zalpha * zzwx + (1.-zalpha) * zzwy )
     !
     DO_3D( 0, 0, 0, 0, 1, jpkm1 )    !-- Tracer advective trend
         pt(ji,jj,jk,jn,Krhs) = pt(ji,jj,jk,jn,Krhs) - ( zwx(ji,jj,jk) - zwx(ji-1,jj,jk  )    &
           &                + zwy(ji,jj,jk) - zwy(ji  ,jj-1,jk  ) ) * r1_e1e2t(ji,jj) / e3t(ji,jj,jk,Kmm)
     END_3D
      ...
 END DO
```

Figure 8 - Code refactoring for managing extended halo in MUSCL advection scheme

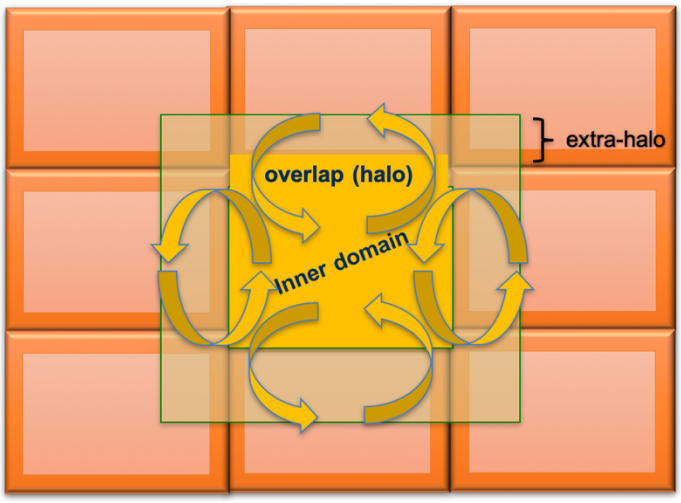

Figure 9 - Extended halo exchange

A wider halo size reduces the frequency of message exchanges whilst it increases the message size at each exchange, as shown in Figure 9. It allows us to adopt some optimisation strategies (i.e. loop fusion, tiling, etc.) to improve the locality.
However, the management of the extended halo size requires addressing the following issues:

1. the management of the existing input files by deleting the global halo
2. changing the code to handle a parametric number of halo rows/columns. These changes impact on both the DO LOOPS indexes within the routines and LBC module, particularly on the north-fold exchanges
3. restart and output files management (read and write operations) in order to avoid saving halo information
4. analysis of the parallel algorithm in each kernel to identify in which routines the exchange can be moved outside by increasing the halo size.

The approach has been implemented and evaluated on the MUSCL advection scheme. The main advantage introduced by a wider halo trades between a reduction of the frequency of the halo exchanges, hence we reduce the communications latency and the "synchronization" points, with an increase in the computational overhead since each process has to compute itself the values itself on part of the halo region. However, the extended halo opens the way for further optimizations that can be applied after moving the halo exchange outside the computational kernels (further optimizations, which are not part of the IS-ENES3 project, include loop fusion or tiling as described in the NEMO development strategy plan). Nevertheless, the use of a wider halo somehow

improves some kernels as in the case of the MUSCL advection scheme; Figure 10 shows the gain in the execution time comparing the original version and the new one with halo extended to 2 lines and the communication moved outside the computing region.

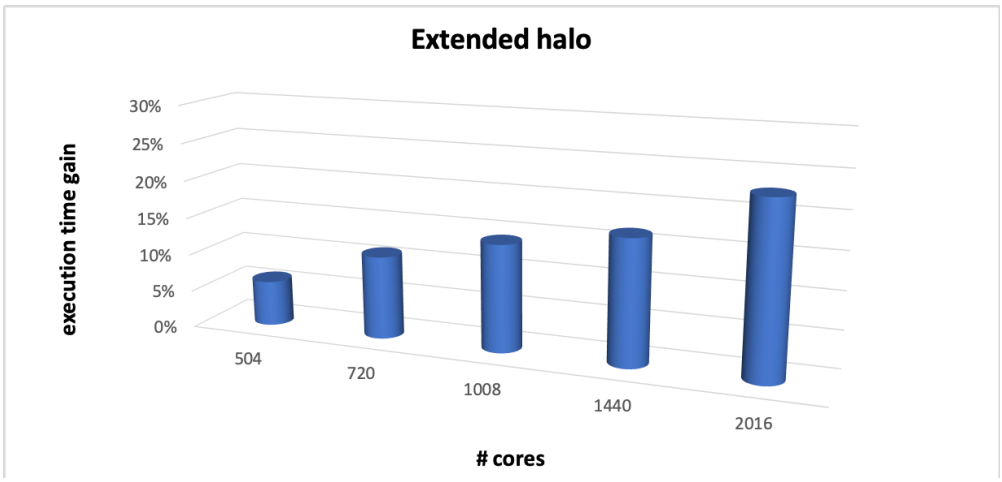

Figure 10 - Execution time improvement (%) of extended halo version compared with original one

2.3. Mixed precision in NEMO code

We present in this section an automatic methodology (auto-RPE) which could be used for any model (included NEMO) to test the precision of the variables, evaluate which ones could reduce their precision and validate the results in scientific terms, through a battery of tests which are executed using a workflow manager.

Auto-RPE: the new methodology to produce mixed precision models

The idea of speeding up computational models by using the least number of significant bits possible has been around during many years. In Baboulin et al. 2009 the authors suggest that by using single-precision (32-bit) floating point numbers instead of the de facto standard double-precision (64-bits), the performance of many algorithms might be enhanced while maintaining the accuracy. In Váňa et al. 2017, it is shown that by means of this approach the atmospheric model IFS obtained an average gain in computational efficiency by approximately 40%. When trying to use the same approach on NEMO we observed that although the performance gains were of a similar magnitude, the outcomes of the simulations were sensibly different. This result suggested that optimizing the numerical precision would be a very valuable optimization on the one hand, whereas on the other hand it would be necessary to first identify where single precision was insufficient to safely reduce the numerical precision used in NEMO. Finally, working only for NEMO to reduce the precision could be a waste of time and not interesting enough for the weather

and climate community, so a general methodology that could be applied to other models seemed to be more convenient.
However, determining which parts of a code are precision sensitive is not straightforward. Relying on expert knowledge to point out the most suspicious regions can be insufficient given that some of the issues can be really hidden and not intuitive. Moreover, it does not represent a robust method which can be scrutinized and reviewed [4]. Instead, we intended to develop a method which could be used to identify sensitive regions even without a deep field knowledge. The idea behind the method in progress is to define when we consider that the results are accurate and automatically find which configuration minimizes the numerical precision used while achieving it.
In summary, the new method on development, AutoRPE (Automatic Reduced Precision Emulator) is a python tool designed to allow the optimization of numerical precision in FORTRAN codes.
It was originally developed to work with the ocean model NEMO, although we aspire to make it usable with other FORTRAN codes. A general sketch of how the tool works is described by the following points:

- Analysis of the pre-processed sources, and storage of all the information about: variables, functions, subroutines and modules used in the code.
- Substitution of REAL variables with custom RPE type.
- Through the coupling with a workflow manager a series of simulations is run, driven by a search algorithm that discriminates between good and bad results. Eventually a list of variables is produced, with information on the precision each of them needs to retain.
- Automatic implementation of the changes needed in the original code in order to have a working mixed precision binary, following the prescription given by the analysis.

The new methodology will use ensemble executions and statistical techniques to ensure that the chaotic nature of climate systems is took into account. Additionally, the collaboration with the main model developers will be mandatory to propose different configurations witch cover most of the variables included in the model.

3. Deviation and difficulties overcome

During the reporting period, there were not any particular deviations which prevented the performance of scheduled activities and the achievement of the main objectives. Instead, some of the designed optimisations that have already been discussed within the NEMO HPC-WG and the NEMO System Team during the first year of the IS-ENES3 project, have

been partially scheduled in the NEMO Workplan 2020, as well as developed and integrated into the NEMO code.
Due to COVID-19 emergency, collaboration among the partners involved in task 8.1 has been performed through online meetings without any particular deviation.

4. Next steps

During the last two years of the project, development and integration actions of the two optimisations on communications will be fully integrated and tested within the NEMO code. In particular, the support to both the new communication strategies has been already integrated, while during the next year the use will be extended to the whole code. The evaluation of the performance improvement will be completed by using the automatic performance tool. Other optimisations of the main bottlenecks (I/O and communications at the north pole) will be addressed.

5. Reference

[1] https://www.nemo-ocean.eu/wp-content/uploads/NEMO_Development_Strategy_Version2_2018-2022.pdf
[2] https://cerfacs.fr/wp-content/uploads/2019/01/GLOBC-TR_Maisonnave-Nemo-2019.pdf
[3] Using MPI, 3rd Edition, by William Gropp, Ewing Lusk, and Anthony Skjellum, published by MIT Press, 2014; ISBN 9780262527392.
[4] Tintó Prims, O., M.C. Acosta, A.M. Moore, M. Castrillo, K. Serradell, A. Cortés and F.J. Doblas-Reyes (2019). How to use mixed precision in ocean models: exploring a potential reduction of numerical precision in NEMO 4.0 and ROMS 3.6. Geoscientific Model Development, 12, 3135-3148, doi:10.5194/gmd-12-3135-2019.