NEMO version 5 computing performance (CPU)

Éric Maisonnave

NEMO version 5 computing performance (CPU)

E. Maisonnave
LOCEAN, CNRS, Sorbonne Université, Paris
January 2025

Abstract

We take benefit of an important milestone in NEMO development (the version 5 release) to update our model performance measurements on two of our current CPU based supercomputers. With BENCH testing configurations and internal timing tools, we verify the highly positive trend of recent developments. In particular, the tiling instrumentation confirms its capacity to speed up computations at memory bound pace of the model (large MPI sub-domains). Even if we cannot find any evidence that OpenMP parallelisation of the tiling loop would have a favourable effect on future similar platform, we strongly advocate for the maintenance as is of the tiling instrumentation

Table of Contents

1- Introduction

We take benefit of an important milestone in NEMO development (the version 5 release) to update our model performance measurements on current CPU based supercomputers. Since 2018, important remodelling of numerics and algorithms have substantially modified the code and improved its performance. Even if the impact of such modifications is constantly checked by the developers themselves, a broader and more detailed picture of this impact is proposed here, in such a way that new enhancements could be proposed on the top or in complement of the existing developments.

1.1- The NEMO scalability paces

We start from the diagnostic that performance of complex model such as NEMO is hard to summarize with only one single number. First because NEMO is proposed and maintained on several configurations which computing needs subtly vary. Second because of the hardware variable capacities on which the model is used. And third because NEMO is a parallel model, which performance changes with decomposition.

Decomposed into local subdomains spread over distributed memory systems, what we propose to call "paces" can robustly be observed side by side on a performance vs resources plot, like on the simplified scalability plot shown on Figure 1 below.

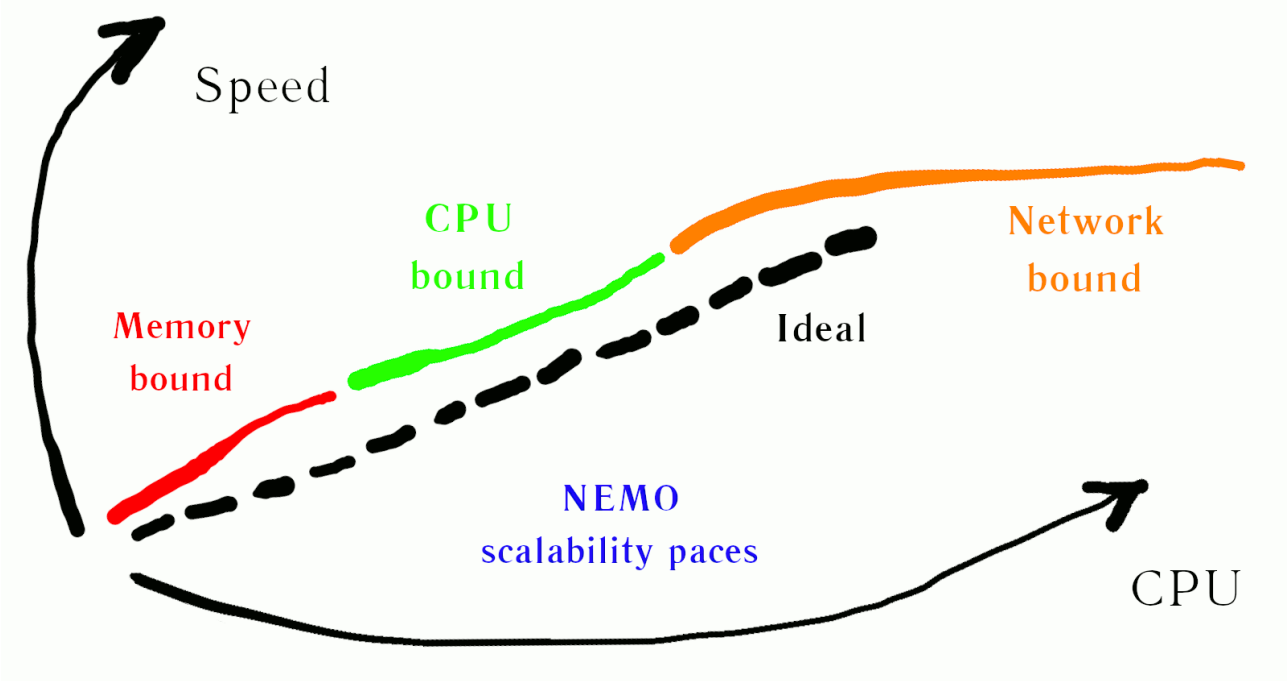

Figure 1: Canvas of NEMO scalability

Modulated by hardware characteristics and model parameters, and independently of any additional IO constraints, a scalability plot of the NEMO model usually shows three distinct paces, following an increasing level of MPI parallelism (i.e. computing resources):

1- Distributed on few resources, scalar computations of large subdomain arrays are mostly waiting data uploads and downloads on a limited amount of memory (**memory bound** pace, in red on Fig 1.) Computation speed per grid point increases with resources, which leads to **hyper-scaling** : a model decomposition on two times more resources goes more than two time faster.

2- The next pace is reached when the subdomains array size is small enough to adequately feed the memory caches, and speed up is then only limited by the amount of resources available (**cpu bound** pace, in green). In these conditions, the **scalable** (also called "ideal") pace is reached.

3- When too many subdomains wait their boundary conditions concurrently, imbalanced communications then limit the speedup and the **sub-scalable** model enters the **network bound** zone (in orange). At the end, most of the time is spent to wait and exchange boundary conditions, and to perform computations on halos larger than their inner subdomain.

A lot of recent research were focused on the network pace limits, by reducing the so called North Folding communication bottleneck, by grouping or removing all unnecessary boundary conditions or collective communications. In addition, the time scheme change (MLF to RK3), allowing larger time step, has greatly limited the needed resources on the scalable pace. And reduction of array size and number, combined with the implementation of a tiling loop, has improved the performance on the memory bound zone. It is clear for us that a comprehensive measurement of the last year performance gain should be done on each pace of the MPI decomposition range, which motivated this larger than usual exercise of measurement.

1.2- BENCH configurations

Such time consuming task could not be done efficiently without a simplification of the measurement protocol. Even limited on global configurations, a full exploration of the model paces requires the use of at least three resolutions (1, ¼ and 1/12 degrees, resp. BENCH-1, BENCH-025 and BENCH-12). The handling of input files, and the time spent to load them, prevent the use of the real (and even not officially supported) configurations for benchmarking. That's why the realistic BENCH configurations were developed [Irrmann et al. 2023] and why they are used in this work.

Some limitations to their realism are well noticed : assuming that the communication pattern has an impact on performance (e.g. by changing the number and ordering of communications thus the communication balancing), the absence of land-only subdomains could become a problem for finer studies. A realistic distribution of land-only sub-domains in BENCH would require to read a land map from a file, incompatible with the portability constraint, or the inline

definition of a simplified land map. The abrupt bathymetry created could also have a negative impact on numerical stability. Such refinement of our tool could be done in a future step if (unanimously) required. Notice that the absence of land-only sub-domain leads to an overestimation of the model speed: for a given number of resources, a real global configuration covers a three times smaller area and is decomposed on three times smaller subdomains.

The same communication pattern realism is questioned by the simplified ice zone originally proposed for SI3 representation. Work presented here scarcely deals with sea-ice model so that the issue could be described (and addressed) in a separated report dedicated to the ESIWACE3 dwarfs project [Maisonnave 2025]. TOP-PISCES biogeochemistry is excluded from our present work, but its structure is not fundamentally different from the rest of the code and would not significantly change the ocean-ice results presented here.

1.3- Target machines

We rely on two supercomputers to quantify our model performance:

- the same that previously used to get the performance of the reference NEMO 3.6 version in 2018 [Maisonnave & Masson 2019]. The extended longevity of french supercomputer has the nice side effect to allow comparison on the same hardware during larger (and larger) periods
- and a more recent machine, with Non Uniform Memory Access (NUMA), allowing groups of eight cores to preferentially access a 32MB memory cache at high speed. This should speed up configurations able to keep data in this quickly accessible memory cache.

A brief is given below on Table 1:

	TGCC 'irene', France	DKRZ 'levante', Germany
CPU, compiler	Intel Skylake, Intel-2020.0.0	AMD Milan, Intel-2022.0.1
MPI library	OpenMPI 4.1.4.5	OpenMPI 4.5.3
Nodes available on prod	1653	2670
Cores&Memory/node	48 - 180GB	128 - 256GB
Cache L3	38.5MB shared	32MB per 8 cores
Network	Mellanox Infiniband EDR100	Mellanox Infiniband HDR100

Table1: Hardware-middleware characteristics of the two Eviden machines of our study

1.4- Timing tools

First, it is important to define what performance really means for geophysical models, e.g. from the quasi-exhaustive list given by [Balaji et al. 2017]. In the present document, we focus on the model speed, defined as the time spent in the inner time loop restitution time, excluding initialisation and termination time. Actual performance, including IO, workflow and machine

scheduling strategy, is not included in our study. Quantification of memory requirements will be made indirectly.

Timings are performed with the NEMO internal tool [Irrmann et al. 2023]. Its main advantages are:

- simplicity of use : no instrumentation, no post-processing,
- portability : no need of additional libraries, even for visualisation which is perform with our usual netCDF based tools,
- standardisation : everybody can speak the same language because measurements are identically and non ambiguously defined,
- and results compactness: capacity to deals with the model parallelism without burying users under tons of useful but sometimes weighty information

In this document, we call model speed one single information provided by the NEMO internal timer: the time spent by the slowest process performing time loop computations between time step `kt000+3*sbc_frequency` and `ktend-2*sbc_frequency` (hereafter "inner time loop"). But sometimes, e.g. to estimate the minimum time spent on dedicated sub-routines (like `lbc_lnk` for the lateral boundary communications) timing of the fastest process will also be considered.

Some options could be particularly well suited for our kind of measures, such as the possibility to disconnect communications (and deduce their impact by comparison)[1]. In this case, the communication impact will be estimated by the difference between the slowest and the fastest process to perform the `n-5*sbc_freq` steps of the (inner) time loop.

We did not find necessary to go further by checking more carefully the behaviour of each of our parallel processes, with more accurate analyser such as `paraver` [Pillet 1995]. However, it is still impossible with such synthetic analysis to finely understand the whole communication pattern execution at time step level, as can be done for example with the OASIS internal measurement tool [Piacentini & Maisonnave 2020]. And it is impossible to statistically but unambiguously determine a "communication time" that could characterise the way our implementation is taking benefit of the MPI library and underlying network capacities.

1.5- Protocol

It is widely known that performance measurement reproducibility is practically impossible to obtain in normal conditions. With its high level of parallelism, performance is particularly sensible to middleware (OS preemption, disk access ...) and hardware behaviour (node heterogeneity, memory access heterogeneity, failures ...) Since our tests were done on a production machine, even adjacent jobs can add perturbations to our measurements.

This is how we tried to reduce the spread of our measures resulting from this variability:

1 Unlike the ocean part, the SI3 still requires collective communications, that have to be manually removed (in `ice_dyn_adv_pra`, for BENCH) if we want to fully free our model from MPI communications

- limit the chance of significant disturbance by reducing the simulation length to 50 (BENCH-12) or 100 time steps (BENCH-025, BENCH-1),
- remove outlier simulations affected by significant disturbance, by checking the time step execution spread[2],
- measurements are performed with magic number decompositions, predefined to minimise perimeter/area ratio (thus the amount of lateral boundary communications), that also minimises the number of unoccupied cores per node.

2- Model performance

2.1- Survey

Model performance is regularly checked during development process, particularly when speed improvements are expected. It was the case this year, as shown below on Figure 2.

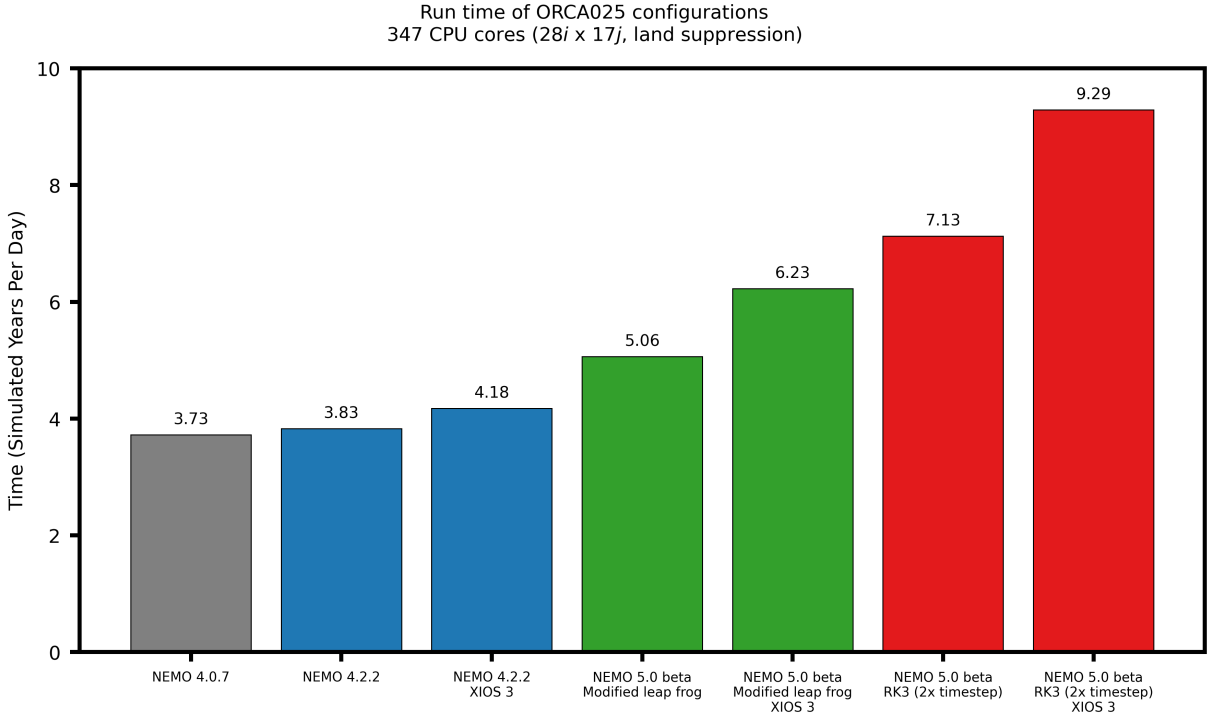

Figure 2: performance of NEMO versions, 2024, courtesy Daley Calvert

Several versions are simultaneously installed on an available machine so that code impact can be directly evaluated, thanks to the git based versioning tool adopted by the NEMO-ST. The working ORCA025 configuration chosen by Daley Calvert requires the maintenance of the successive input files, but actual model results are directly applicable for users real cases. Figure 2 results show a large impact of recent time scheme change and various other

2 `timing_step` variable from the corresponding netCDF timing output file `timing_step.nc`

improvements (x2 from 4.0.7 version to 5.0 beta). The ORCA025 resolution/low resources association suggests that NEMO is operated here on its hyper-scaling pace.

For the reasons explained above, the BENCH test configuration was better chosen to lead our study. We directly compare our results with a measurement set taken during a performance improvement exercise, on the same target machine [Maisonnave & Masson 2019]. The code evolution encompassed here is larger (from 3.6 modified leap frog to 5.0 Runge-Kutta 3rd order) and model resolution is smaller.

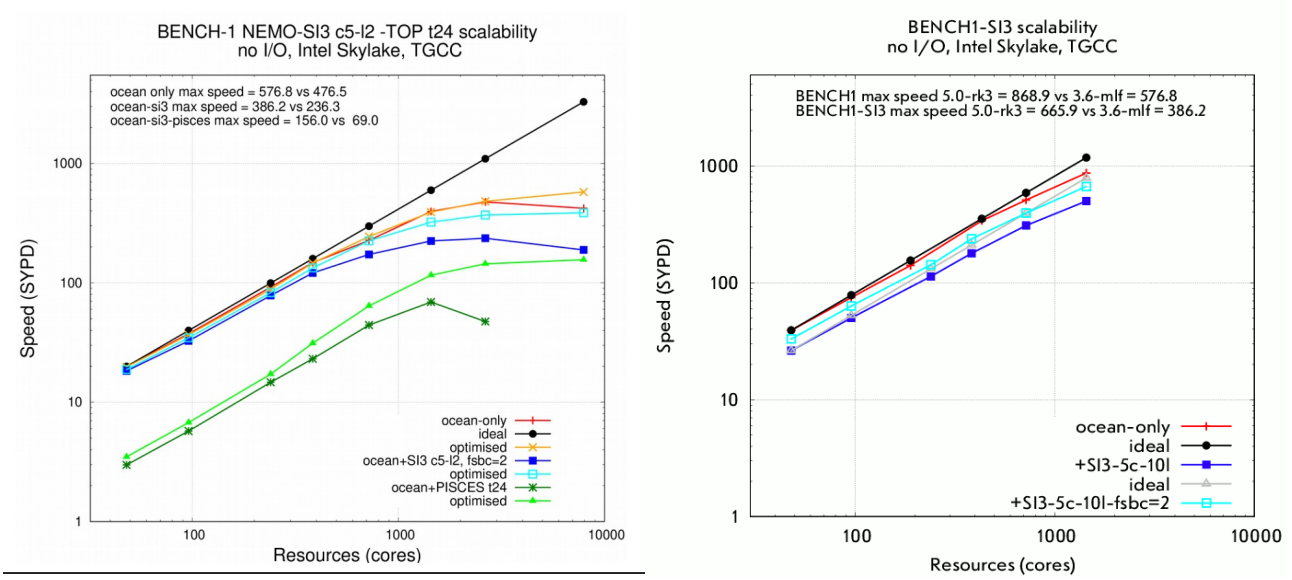

Figure 3: BENCH1 scalability on the same Intel Skylake `irene` supercomputer, with NEMO3.6 (left) and NEMO5 (right). Ocean-Ice simulations can be directly compared with coupling time step period = 2 (cyan curbs)

But, as we see comparing Figure 3 left and right, if we focus on the memory bound pace (low resources) speed up can be also estimated to x2, for both ocean-only and ocean-ice configurations. At scale, the maximum speed is also close to be doubled. As shown on Figure 4, with different units[3] and on the DKRZ machine, this maximum is reached for a 8x8 size of MPI-decomposed subdomains.

3 Subdomain size, inversely proportional to the root mean square of the resource # and time to solution, inversely proportional to speed

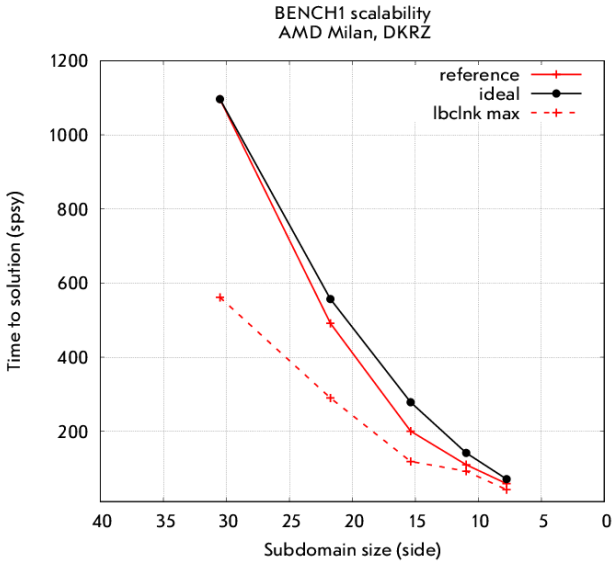

Figure 4: same BENCH-1 scalability than Fig.3 (NEMO5, right), but different axis. Root mean square of sub-domain size (instead of resources) and time to solution (instead of speed). Scalability of lateral boundary condition routine (max from all processes) is in dashed line

2.2- Can we evaluate a "communication time" ?

After a quick look at the routine timing ranking, it seems easy to consider `lbc_lnk` (lateral boundary communication routine) as the major bottleneck, particularly at network bound pace but not only. On the same Fig. 4, we superpose to the total time the time spent in this lateral boundary condition routine (including all MPI communications of our BENCH configuration). The maximum values (between processes) of this quantity gathers half of the total time on memory and CPU bound paces, and even more at scalability limit. This number deserves a detailed analysis, assuming that intra- or inter-node communications cannot be per se that time consuming, particularly at parallel scales which are not supposed to use that intensively the communication network.

Figure 5 (left) represents the total `lbc_lnk` timing for each sub-domain in a 2D global array (Northernmost processes on the top, Easternmost on the right). The slowest processes, responsible of the high ranking of the routine, are clearly located in the North, which is coherent with the automatic decomposition of the grid: 45x16 grid points are allocated to these processes, instead of 45x21, to compensate the slowest communications executed on the region of the North Pole Folding (NPF). From this, we deduce that the maximum values of `lbc_lnk` timing are reached when the least loaded processes (on NPF) are waiting for the others.

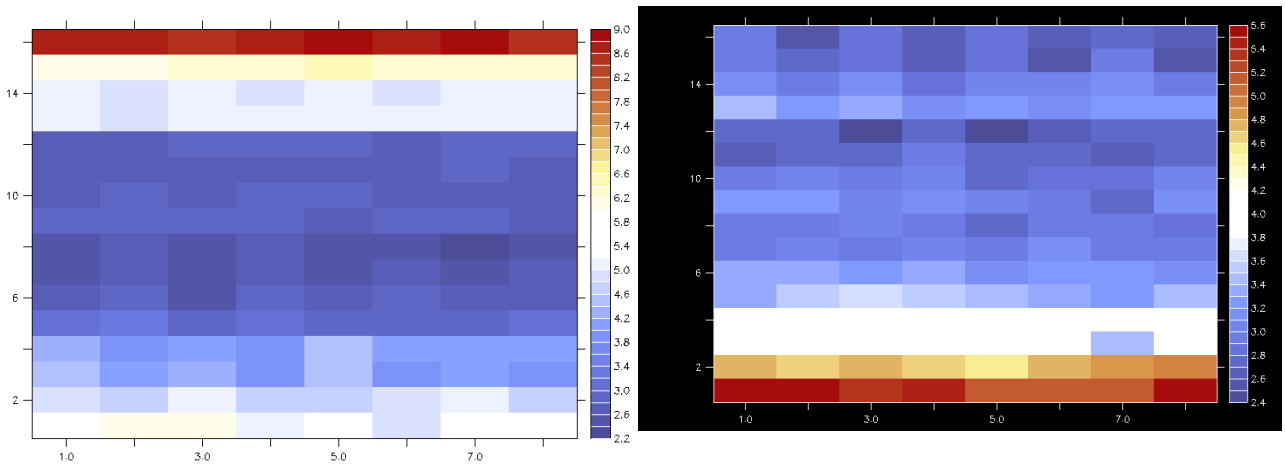

Figure 5: `lbc_lnk` timing per MPI process, in reference run (left) and balanced configuration, with same communication pattern, same 128 resources but balanced number of grid point per process (right). Time in second is measured in the inner time loop

However, timings on the other processes are not equal. This can be confirmed with an easy modification of the BENCH parameters[4], by changing MPI decomposition thus total grid point # setting equally loaded sub-domains (with 45x21 grid points, in the present case). We call "balanced" this testing configuration. On Fig. 5 (right), the largest values are indeed disappearing from the North raw: the largest values observed in the reference run were mainly due to load imbalance and not to extra communications in these areas. We put in evidence, with this simple test, that `lbc_lnk` timings include both communication and waiting times. Depending on the relative contributions to these two, the problem on how to speed up `lbc_lnk` will be addressed differently.

Our balanced configuration still exhibits a maximum `lbc_lnk` timing of 5.6s on some processes and a spread of 3.1 seconds. The highest values are now located on the Southern part of the grid, where only five lateral boundary conditions are exchanged (instead of eight). Once again, we suppose that waiting times are at the origin of these Southernmost maximum values.

To confirm this hypothesis, we switch off any lateral boundary condition communication[5]. On Figure 6, the same restitution time per subdomain is shown, but this time for the total time loop (inner part). Obviously, without actual MPI boundary condition exchanges, the `lbc_lnk` timing is now close to zero[6], the computation load imbalance accumulates at each time step and can finally be measured at the end of the time loop by comparing the total time spent on the loop for each (computationally independent) subdomain.

4 With negative value of `nn_isize` & `nn_jsize` namelist parameters, we set subdomain length and width instead of total dimension of the global grid

5 With namelist parameter `nn_comm=0`. Notice that with the sea-ice model (not involved in this study), an additional neutralisation of a delayed global communication is necessary to deduce the computational load imbalance with this method

6 Due to the implementation of the `nn_comm=0` special configuration, even though LBC MPI communications are neutralised, `lbc_lnk` is still called, communication arrays still copied and only MPI API subroutine calls avoided. This let measuring array copies in `lbc_lnk` (0.4 seconds in our case)

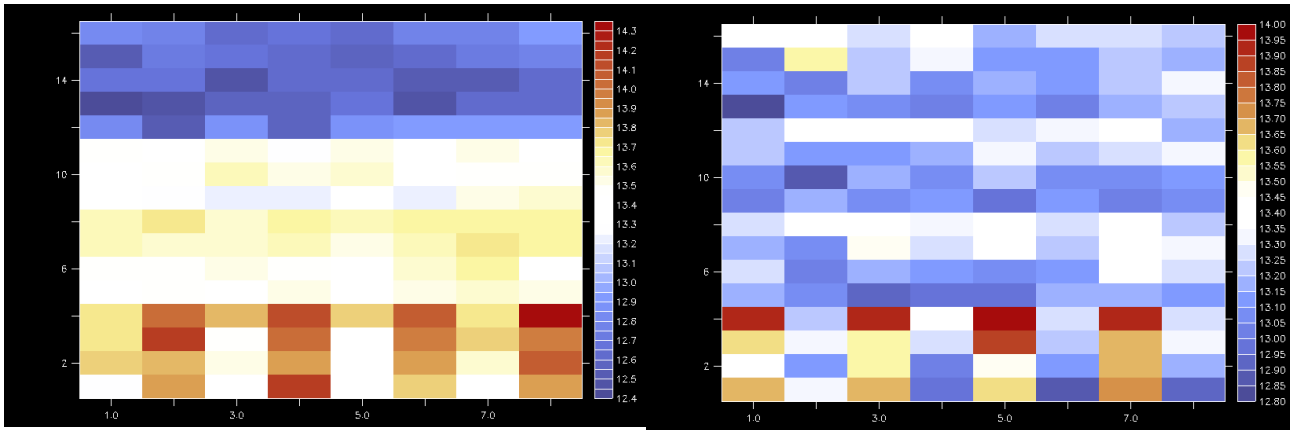

Figure 6: time spent on inner `stp` time loop, `LBC` switched off, on reference (left) and balanced (right) configurations

In the reference run with real unequal subdomain sizes (Fig 6. left), faster subdomains are located in the northern part of the grid. This can be again explained by the automatic decomposition strategy followed by the model algorithm in case of no LBC communication[7]. Computations on the smallest subdomains end first. On largest subdomains, they end with a delay of 1.8 seconds. However, it looks obvious that all the largest subdomains do not end at the same time. This can be confirmed by using the balanced configuration on the same communication-less mode (Fig 6, right). Although all subdomains are now independent and performing exactly the same amount of communications, the spread on total computation times still reaches 1.1 seconds. Interestingly, the same picture obtained on another computing node during a second experiment shows a move of the slowest subdomain (Figure 7a). A third experiment shows the slowest subdomains on a new location again (Figure 7b). But during a fourth experiment, performed just after the first model execution on the same node and the same job (Figure 7c), slowest domains stay at the same location. This illustrates that the spread in computation times has its origin on hardware or middleware heterogeneity.

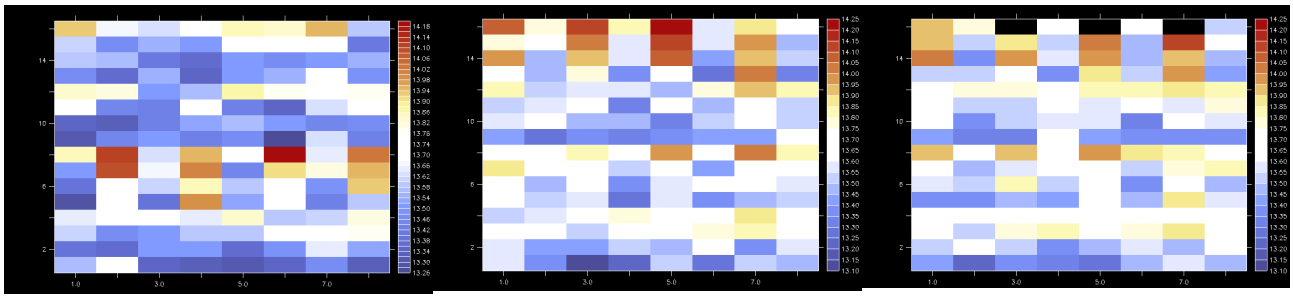

Figure 7: time spent on `stp` time inner loop, `LBC` switched off, on balanced configuration, different execution of the same experiment. The middle and right plots display two experiments launched sequentially on the same node during the same job

7 All subdomains must have the same dimensions and the division rest of the total domain size by the X and Y decomposition must be subtracted one by one from the Easternmost/Northernmost subdomain dimensions

To summarise, we could separate the total NEMO simulation time on one node of our machine into five pieces (plotted with different colors on Figure 8):

- what can be called a "computation time", defined as the minimum time spent by the fastest subdomain in a balanced and communication-less configuration,
- a "machine load imbalance" provoked by hardware and middleware heterogeneity and measured as the computation time spread in a balanced and communication-less simulation,
- a "static boundary communication time", defined as the time spent locally copying arrays in `lbc_lnk` routines,
- a "communication and load imbalance effects", defined as the minimum time spent in `lbc_lnk` during a balanced simulation, minus the last two timings,
- a "load imbalance cross effect", defined as the remaining time when the four previously defined timing are subtracted from the total reference time.

None of these five contributions can be purely called "communication time". In a lack of more detailed analysis of each send/receive event (as it could be done with a tool like `paraver` and a comprehensive statistical analysis), we can only deduced an upper boundary of this MPI library effect on our performance from the addition of the last two numbers.

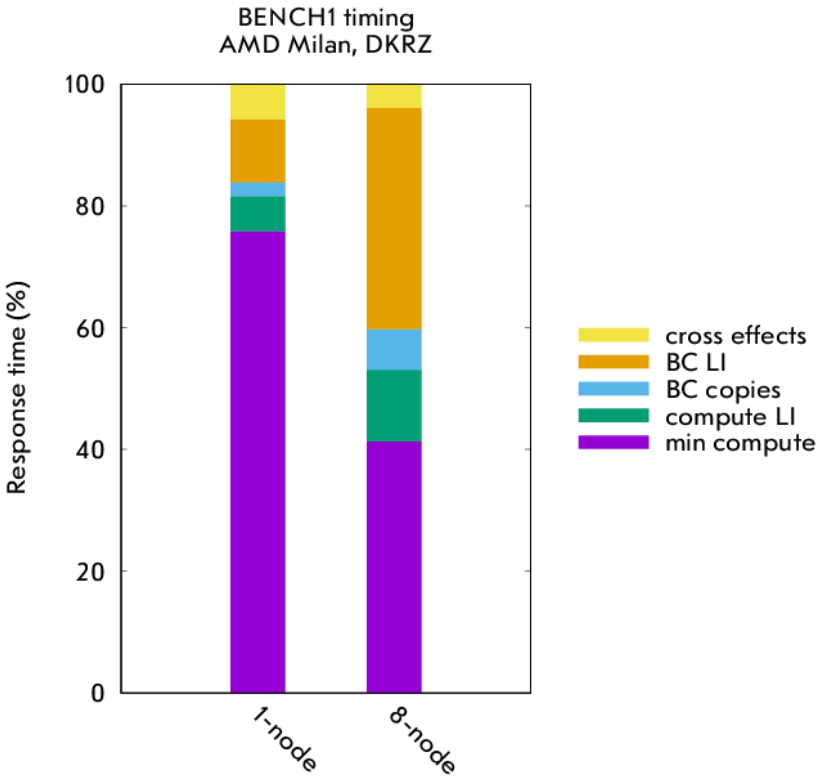

Figure 8: Total reference simulation time split into different contributions (%) on one and eight nodes of our supercomputer

The same exercise (reference, balanced and communication-less simulations) is performed on eight nodes of our machine. We show on Figure 8 the increase of the machine (in green) load imbalance but more significantly of the communication load imbalance (in orange). The picture of load imbalance per sub-domain (Figure 9) lets appearing the mapping configuration of MPI processes[8], and reveals machine or middleware slowing down on most of the running cores.

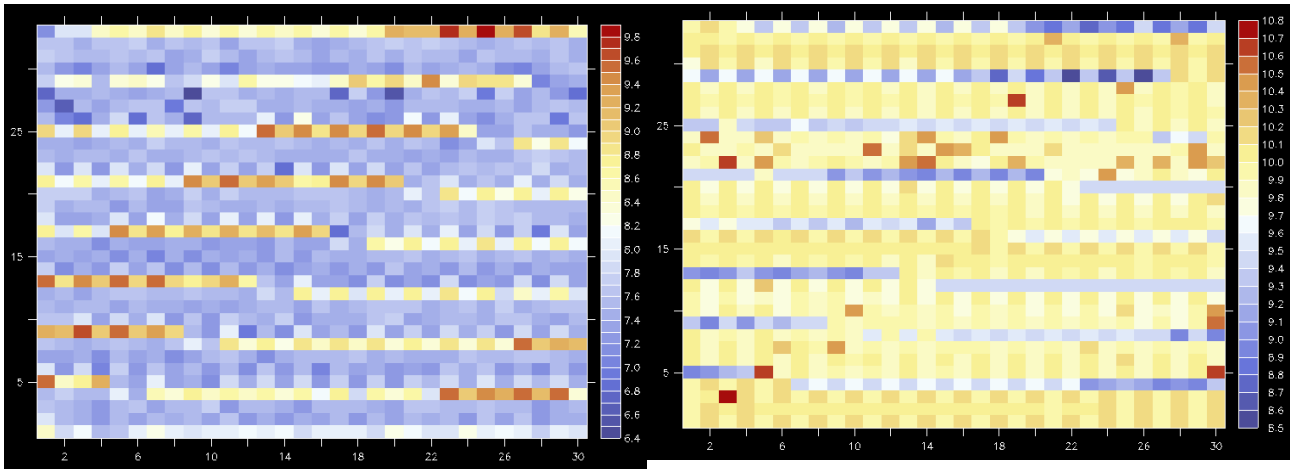

Figure 9: on eight nodes, `lbc_lnk` timings with a balanced configuration with LBC on (left) and `stp` timings with the same balanced configuration but without LBC (right)

This set of experiment helps to show that, despite alarming timing values of the communication routine `lbc_lnk` when profiling a reference simulation of NEMO, at any model pace, the real part of MPI communications is generally small. It can be estimated at its maximum to only less than 40% of the total time, on eight nodes of our machine, when subdomain size reaches the value 12x11. More importantly, we show that a set of specially configured simulations must be launched to be able to decompose load imbalance into separated contributions (hardware+middleware, computing, communications) that affects simultaneously the real computations.

Such (costly) experiment would have to be repeated with several other configurations (resolution, ice and BGC modules) and on different platforms to fully characterise the NEMO version 5 performance. However, we consider that we can now rely on a reference to keep exploring and quantify the effect of other model HPC past or new related developments, such as tiling.

8 Slurm option of srun: `--distribution=block:cyclic`

3- Tiling

At NEMO memory bound pace, any algorithm modification that would reduce the size of the working arrays would necessarily speed up the model by lowering the amount of data continuously transferred from memory to registers. In that perspective, Calvert 2024 has organised at the highest possible level the splitting of local subdomain arrays into tiles treated sequentially.

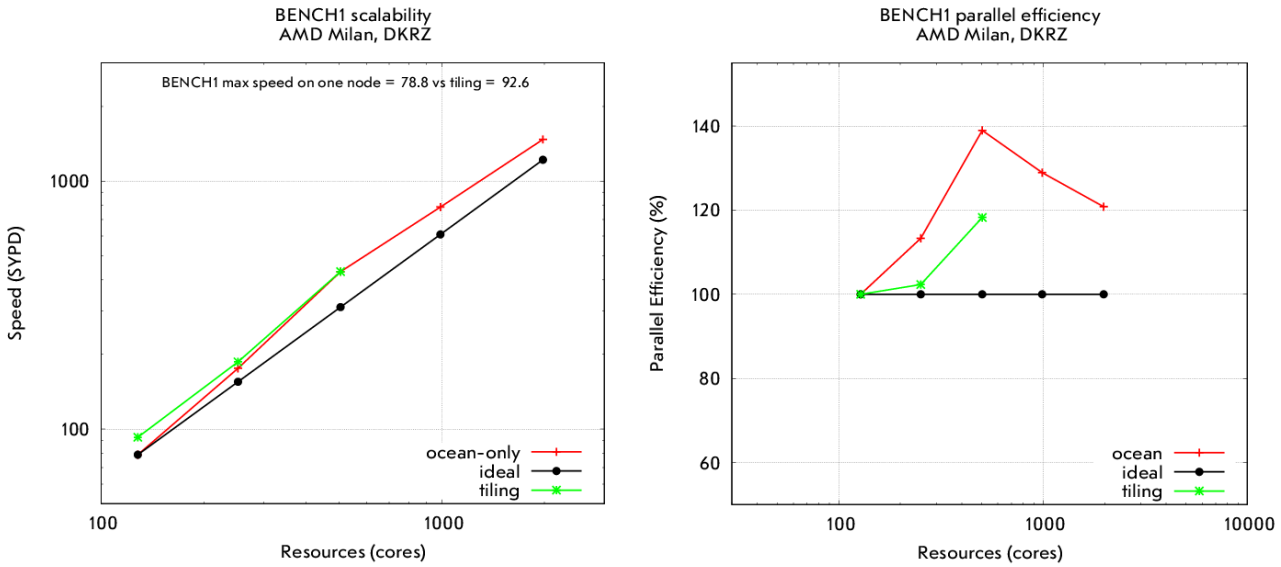

Figure 10: BENCH-1 ocean-only scalability plot with (green) or without (red) tiling, represented by speed (left) or parallel efficiency (right)

We decide to activate tiling in our setup[9], and explore the range of tiling parameters (X and Y splitting directions[10]). It quickly appears that X direction could not efficiently be used (probably conflicts with vectorisation). On the measurements shown below, only the best value of Y tile length has been displayed[11]. In extreme cases, the lenght value 1 gives the best performance, which means that tiles with four grid lines halos et only one computationally effective line outperform the non tiled reference algorithm.

Unsurprisingly, with BENCH-1 ocean-only, on the memory bound pace of the MPI decomposition range (low resources, larger subdomains), the tiling has a comprehensive effect and helps to reduce hyper-scaling by about a factor 2 (see Figure 10). These good results are not surprising on such platform where quick cache buffers are attached to subsets of computing cores (see Table 1). Tiling enhance performances of most of the model routines[12]. A closer look to some accelerated routine like `zdf_phy` shows better performance on every sub-domain, with a better load balancing. However, hyper-scaling cannot be totally

9 `ln_tile = .true.`

10 `nn_ltile_i` and `nn_ltile_j`

11 Interestingly, the best value is always at the maximum of a parabolic curb

12 except `eos_pot, bn2, rab_3d, eos_insitu` and `tra_adv_trp`

removed, which suggests that tiling does not reduce the memory footprint so that all working arrays could fit in the quickest buffers of the processor.

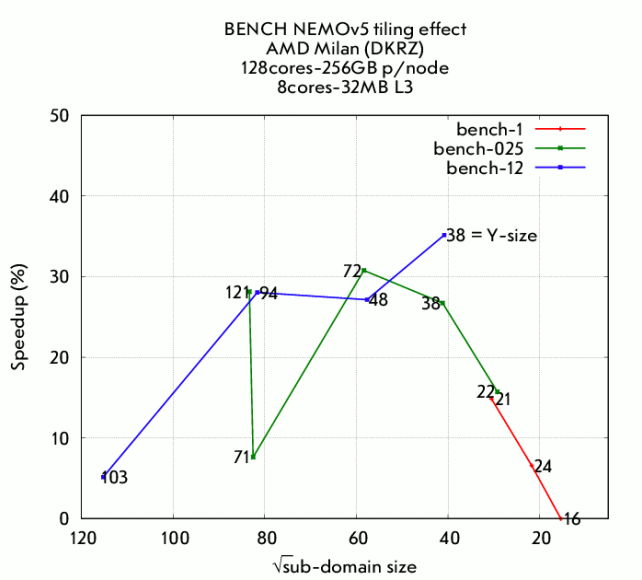

Figure 11: speedup gain (%) of tiling compared to reference algorithm (default values namelist) at 1, ¼ and 1/12 global resolution (BENCH configuration)

Extra tests are performed with BENCH-025 and BENCH-12 resolution, confirming the beneficial effect of tiling with even more memory bound configurations. On Figure 11, the benefit of tiling is plotted for three BENCH horizontal resolutions, as a function of the root mean square of the subdomain area (in grid point #). A maximum of 30% is reached for 60x60 subdomain size. Tiling is efficient on large parts of the resources range, and this is more and more true with increasing resolution. As an additional information, the actual Y size of the subdomain is displayed on Fig. 11 beside each measurement point, to add a crucial information for tiling: the maximum number of lines that can be split to create tiles. As expected, subdomains presenting a Y side larger than X side exhibit better ability to benefit from tiling, which suggests that an extra constraint should be added in the decomposition choice to take into account tiling potentiality.

A cross validation from MetOffice on AMD Milan and Genoa processors is planned to confirm or invalidate these encouraging results.

For the moment, these results advocate for keeping the tiling instrumentation in the code, at least for users who usually lead NEMO simulations on limited amount of resources and with high load per process unit (at memory bound pace, which seems to be dominant in our set of survey experiments). If we believe that the future of CPU computing will come with more CPU per node, tiling deserves to be maintained and continuously validated. This arises a set of limitations/difficulties:

1. the tiled domain overlap (halos) is error prone for developers. None of the arrays modified in the loop cannot be reused for spatial derivative computations before the end of the loop (which is rightly done until now)
2. tiling does not cover the entire code. Particularly, we are missing XIOS calls, and even more significantly, ice & BGC models
3. the working arrays size reduction is known to be counter productive for GPU (& vectorisation, see X-direction inefficiency for tiling)
4. since this tiling feature is totally inefficient at high resources, it rather addresses Low Carbon Computing (LCC) issues than HPC.

4- Discussion

This special focus on model tiling capability was guided by the idea of summarising NEMO version 5 performance but also to explore new ideas of performance improvements. This work check the current ability of our code to perform well on the current CPU platform but also makes the assumption that additional performance on this kind of machine is desirable in a near future. This does not suppose that coding must ignore GPU compatibility but we propose here not to definitely act the removal of any computing oriented coding, as soon as automatic instrumentation of a supposed computer agnostic code would not have shown its superiority.

4.1- Theory of hybrid MPI/OpenMP performance

Following this philosophy, we examine the idea to rely on the existing and non intrusive implementation of tiling to add one level of parallelism in NEMO with OpenMP. The idea has already been explored at MetOffice (Ganderton, 2023). Tiles are grouped in OpenMP threads and computations performed in parallel rather than sequentially.

The idea has numerous advantages (existing coding, parallelism that also partially keeps tiling benefits) but has two main limits :

- Mutually exclusive MPI/OpenMP parallelisation for a given number of resources. To be efficient, the OpenMP implementation supposes a better efficiency than the corresponding MPI one. Said differently, if we name restitution time "`T`", the % of time spent on OpenMP section "`x`" , and the # of OpenMP threads "`N`", Amdahl's law for both MPI and OpenMP sections gives this formula for OpenMP efficiency :

$$T > (1-x) \cdot T \cdot N + x \cdot T/N$$, equivalent to

$$x > N / (N+1)$$

Which means that more than `N/(N+1)` part of the code (as measured in flat MPI mode) must be spent within OpenMP sections, if we want to hope a positive speedup with a mixed MPI-OpenMP parallelisation (on `N` threads) compared to a flat MPI[13]. From this we can deduce that OpenMP parallelisation on top of tiling loops would benefit from additional code instrumentation. In addition, as already emphasised by Ganderton, any break between the OpenMP/tiling loops implies synchronisation and loss of performance. Unfortunately, stops are unavoidable at least for MPI communication[14]

- Mutually exclusive tiles/OpenMP parallelisation. If tiling loop were instrumented, more OpenMP threads (`N`) would mean less tiles (`M`). In addition, OpenMP+tiling loop are limited to Y axis so that `N.M` < a few dozens, depending on resolution and total resources. It means that on memory bound paces (most of the time, on our machine), OpenMP would not only have to be more efficient than flat MPI but to be also more efficient than flat MPI + tiling.

Practically speaking, an OpenMP parallelisation on top of the tiling loop could differ depending on the model pace:

- Hyperscaling and scaling paces : OpenMP on top of the existing tiling loop should be combined with tiling, still efficient at this pace. Shared arrays would have to be copied on local private arrays, to avoid race conditions and no "false sharing". But additional memory and synchronisation would be required, jeopardising the good bufferisation effect of tiling. We can already suspect that a better scalability than the flat MPI configuration would be hard to achieve, because of load imbalance and synchronisations added by OpenMP. In practice, as suggested on Figures 10 and 11, these paces are dominant (until 15x15, at least on our machine)

- Network bound pace : only reached for a large number of resources and high resolutions. In this case, tiling would not be needed anymore, and could entirely be replaced by OpenMP (simpler shared arrays synchronisation procedure, no need of overlapped regions). The hybrid MPI/OpenMP model could be more efficient than the existing MPI parallelisation … but the question remains on how to practically decompose our grids on below 10x10 subdomains, to say nothing of side effects on other unexplored constraint of real computations (I/O, scheduling constraints)

From these assumptions, we suppose that in most of the cases, the existing tiling would be more efficient that hybrid parallelisation. However, considering the high relevance of high resolution / highly computing resource demanding configuration popularity among NEMO

13 This assumption can be too strong on communication bound paces of the model, but these paces does not appear to be dominant in our current performance survey

14 Unless multi-threaded MPI communications would be tested, with a strong warning for portability and performance

community, or at least a distinguished part of it, we propose to further explore the few cases were hybrid parallelism could be beneficial.

4.2- Experimental setup

To give a better idea of the challenge, three different experiments are performed on our AMD Milan computer and the timings are combined to try to guess the best MPI/OpenMP performance that we could expect on the present machine. The BENCH-025 configuration (with balanced domains as defined in § 2.2) is launched on 32 nodes of our 128 core per node machine. In a reference run, the subdomain size reaches 22x16 and performance comes close to the so-called communication bound pace, supposed to be the most favourable to hybrid parallelisation. Two measurements are displayed on Table 2: the total time spent in the inner time loop (as defined previously, in balanced mode) and the "communication and load imbalance effects" (supposed to include most of the MPI communication timing, see also measurement protocol at § 2.2).

Experiment	Reference	Depletion (block:cyclic)	Double depletion	Best Theoretical OpenMP
Processes	4020	1024	1024	1024+4OMP
Nodes	32	32	32	32
Sb-domain size	22x16	45x38	22x16	22x16x4
Total restitution time	2.10	14.8	2.10	3.05*
`lbc_lnk` min (best LI+comm)	0.50	1.45	0.49	1.45

Table 2: Theoretical hybrid performance of BENCH-025 on AMD-Milan based supercomputer, given timing of total balanced simulation and proxy to MPI communication timings, on a set of 3 different experiments

This first reference simulation gives the performance we are reaching with an MPI flat parallelisation. We deduce from total and "communication" times that pure computing takes approximately 1.6 seconds at this decomposition. These numbers are confirmed by a verification simulation called "double depletion" on Table 2, in which subdomains of the same size (22x16) are allocated on ¼ of the processors, but using the same number of nodes (32). This configuration requires a reduced number of LBC communications, but of the same size and using the same network bandwidth than in reference simulation. Communications are performed at the same speed and computations are also done as quickly. We can deduce from these two experiments that ¼ of the MPI processes would compute at the same speed on 22x16 sized domains. What now if four additional OpenMP threads would perform these computations in parallel ? With a perfect OpenMP scaling, computations would again happen in 1.6 seconds. However, MPI exchanges would now involve four times bigger LBC communications. In a third simulation, called "depletion" in Table 2, we estimate the time needed to exchange such LBC by performing computations of four times bigger subdomains on the same 32 nodes. "Communication" timings are here evaluated to 1.45 seconds. In theory, the total cost of an hybrid MPI/OpenMP on 32 nodes using four OpenMP threads can be

evaluated to 1.6+1.45 = 3.05 seconds, which is bigger than the flat MPI performance on 32 nodes (2.1 seconds), this without taking into account an additional positive effect of tiling in flat MPI mode.

This result is explained by the relatively good scaling of MPI related timings. In these conditions, it is more interesting to reduce the subdomain size by adding more MPI processes than keeping the same subdomain size for communication and reduce its size by spreading computations on several OpenMP threads: computations are at best performed as quickly but communications are necessarily done at lower speed. A case study where MPI communications would not scale anymore (or anti-scale) is out of reach on our machine. In conclusion, as far as we can know after this study, it is not possible to recommend an OpenMP parallelisation on top of the tiling loop.

4.3- Other solutions

A workaround of the mutual exclusion of MPI and OpenMP parallelisation in our model could be proposed by selecting only parts of the code for OpenMP parallelisation and keeping the other parts in flat MPI mode. Given the difficulty to instrument our code with both MPI and OpenMP, a solution could be to split tasks on separate executables, differently parallelised, and add a macro-task parallelism level for even better performance in concurrent execution.

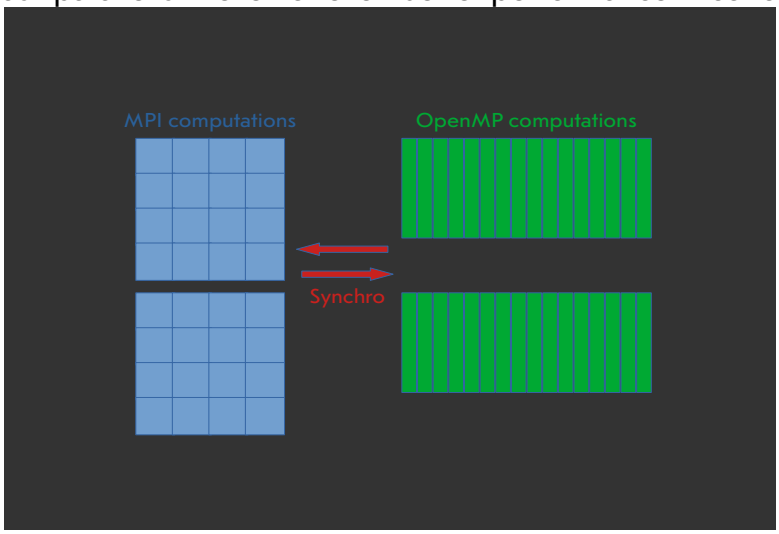

Figure 12: Macro-task parallelism involving several parallel libraries

In the framework described on Figure 12, part of the code taking better benefit of one or the other parallel library would be computed on separated executables and dedicated resources. Of course, OpenMP parallelisation would be limited on one shared memory node, but one can imagine that communication intense routines such as 2D time stepping or ice advection would massively benefit of the disappearing of any LBC routines, that could compensate the lack of parallelism on one single node. This strategy would necessary rely on a fast synchronisation mechanism. For performance, the cost of one input/output gather/scatter communications would necessarily be comparable (if not lower) with the cost of `n` LBC exchanges (and sequential computations). Notice that this strategy would also have a promising GPU declination, particularly on recent unified memory APU infrastructures.

As a side study regarding OpenMP parallelisation, we propose the modularisation and OpenMP parallelisation of one of our network bound routine on whole global domains. For example, the existing TSUNAMI test case, including the 2D time stepping, could be OpenMP instrumented from a starting version excluding MPI. In a second step, this module could be coupled to the complementary NEMO model, to better evaluate the cost of realistic communications (gather/scatter), taking also into account the benefit of the organised asynchronism of our computations.

To summarise, the definition and deployment of a NEMO version 5 benchmarking on the NUMA DKRZ supercomputer has put in evidence the good potentiality for performance of the existing tiling optimisation. However, and independently of the model performance paces at which it would be operated, it looks difficult to overcome the flat MPI and even more MPI+tiling performance with any hybrid MPI-OpenMP parallelisation built on the top of the existing tiling decomposition. The only optional path that we could recommend in the perspective of OpenMP instrumentation is the modularisation of communication intense parts of the code such as 2D dynamics.

Our study has also put in evidence the necessity to check model performance at any pace (memory, compute or network bound), as it is obvious that beneficial optimisations on one part of the parallel spectrum could be totally counterproductive on other paces. This could perfectly lead to the necessity to maintain several codes fitting different needs and different architectures. Code processing tool such as PSyclone could be at this stage a valid option to simplify this maintenance.

ESiWACE3 is funded by the European Union. It has received funding from the European High Performance Computing Joint Undertaking (JU) under grant agreement No 101093054. “This work used resources of the Deutsches Klimarechenzentrum (DKRZ) granted by its Scientific Steering Committee (WLA) under project IDs ka1436 and bk1472 - HANAMI". It also accessed HPC resources of TGCC under the allocation 2025-gen13051 made by GENCI. The author wish to acknowledge Thomas Williams and Colin Kelley for the development of the Gnuplot program, which analysis and graphics are displayed in this report, in addition to graphics from Matplotlib, a Sponsored Project of NumFOCUS, a 501(c)(3) non profit charity in the United States. We acknowledge the use of the Ferret program for analysis and graphics in this report (Ferret is a product of NOAA’s Pacific Marine Environmental Laboratory).

References

Balaji, V., Maisonnave, E., Zadeh, N., Lawrence, B. N., Biercamp, J., Fladrich, U., Aloisio, G., Benson, R., Caubel, A., Durachta, J., Foujols, M.-A., Lister, G., Mocavero, S., Underwood, S. & Wright, G., 2017: CPMIP: Measurements of Real Computational Performance of Earth System Models in CMIP6 , *Geosci. Model Dev.*, **46**, 19-34, doi:10.5194/gmd-10-19-2017

Calvert, D., 2024 : Tiling in NEMO 5 and beyond, NEMO Party, May 17th 2024, Southampton, UK, https://drive.google.com/file/d/1CNPc86nmZh5LtTVrxQnSKqXrKL9vBKy_/view

Ganderton, J., 2023 : Exploratory implementation of OpenMP into tiled regions of NEMO, Student Report, MetOffice, UK

Irrmann, G., Masson, S., Maisonnave, E., Guibert, D., & Raffin, E., 2022: Improving ocean modeling software NEMO 4.0 benchmarking and communication efficiency , *Geosci. Model Dev.* , **15**, 1567–1582, doi:10.5194/gmd-15-1567-2022

Maisonnave, E., & Piacentini, A., 2019: Performance evaluation of the hybrid interactive placement with HIPPPO of SCRIP interpolation tasks , Working note, **WN/CMGC/19/94**, CECI, UMR CERFACS/CNRS No5318, France

Maisonnave, E., & Masson, S., 2019: NEMO 4.0 performance: how to identify and reduce unnecessary communications , Technical Report, **TR/CMGC/19/19**, CECI, UMR CERFACS/CNRS No5318, France

Maisonnave, E., 2025: NEMO dwarfs for parallel optimisation, to be published

Piacentini, A., & Maisonnave, E., 2020: Interactive visualisation of OASIS coupled models load imbalance , Technical Report, **TR/CMGC/20/177**, CECI, UMR CERFACS/CNRS No5318, France

Pillet, V., Labarta, J., Cortes, T. and Girona, S., 1995: Paraver, A tool to visualize and analyze parallel code, *Proceedings of WoTUG-18: transputer and occam developments,* Vol. 44, No. 1, pp. 17-31