Research Papers
Issue RP0109
July 2011

Scientific Computing and Operation (SCO)

NEMO BENCHMARKING: V.3.3 VS V.3.3.1 COMPARISON

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

Silvia Mocavero
CMCC
silvia.mocavero@cmcc.it

Alessandro D'Anca
CMCC
a.danca@vargroup.it

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

SUMMARY The report describes the activities carried out within the NEMO Consortium commitment. They refer to the performance evaluation and the comparison of the NEMO version 3.3 and the NEMO version 3.3.1. The main differences between two versions are related to the memory management and the allocation of the data structures. The NEMO ver. 3.3.1 replaces the static memory array allocation with a dynamic one. This approach brings some relevant benefits, such as the run time evaluation of the best domain decomposition. On the other hand, the dynamic array allocation introduces some lose of computational performance. The aim of this work is to evaluate the difference between the two versions from a computational point of view.

Keywords: NEMO, Performance evaluation, Simultaneous Multi Threading, Domain Decomposition

JEL: C61

This work has been carried out with the support of CMCC-ANS Division, namely Marcello Vichi. 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.

INTRODUCTION

Since the beginning, the NEMO model was designed taking into account the target machine and the available system software. The static allocation of the data structures employed in the NEMO model up to the version 3.3 is justified by several reasons:

1. The optimizations, made by the compiler, can be more incisive when the dimension of the array are known at compile time, like for example the optimization regarding the loops.
2. The allocation of the memory is performed once during the initialization of the data structures and it remains unchanged during the execution of the model.

In a dynamic computing environment where the available resources can change rapidly, the static memory management could be a limiting factor for an efficient use of the resources. Moreover the static approach introduces other related limitations:

1. The domain decomposition must be chosen statically before compiling the code.
2. If the number of available computing resources changes during a long experiment, a re-compilation of the code is needed.
3. The domain decomposition must be maintained static for the whole duration of the run.

The version 3.3.1 of NEMO implements the dynamic allocation of the data structures. This work aims at evaluating the differences between both versions from a computational point of view.

DESCRIPTION OF THE TEST CASES

The comparison of both versions has been performed taking into consideration the following configurations:

1. ORCA2LIM_PISCES
2. POMME
3. MFS16

The first two configurations belong to the NEMO distribution package. They do not represent meaningful configurations from a scientific point of view but they are used as test cases for all of the modification and updating to the model. The MFS16 configuration [2] instead is a production configuration designed at INGV (Italy) with the aim to study the Mediterranean Sea.
The comparison aims mainly at:

1. Evaluating the performance of the dynamical memory version of NEMO against the previous version comparing the parallel execution time, the parallel scalability and the memory size used.
2. Evaluating of the performance both on IBM PWR6 and on NEC SX9.
3. Estimating the efficiency of the domain decomposition algorithm.
4. Evaluating the impact of the SMT (only for IBM).
5. Evaluating the exclusive allocation of computational nodes against a shared nodes allocation.

Accordingly with the previous goals, the test plan described in Tab.1, Tab.2, Tab.3 has been designed:

Table 1: Test plan with MFS16 configuration

Test ID	Arch	NEMO ver.	SMT	Exclusive Mode	Dom. Decomp.
T01	IBM	v3.3	ON	ON	Auto
T02	IBM	v3.3.1	ON	ON	Auto
T03	IBM	v3.3.1	OFF	ON	Auto
T04	IBM	v3.3.1	ON	OFF	Auto
T05	IBM	v3.3.1	ON	ON	Manual
T06	NEC	v3.3	-	ON	Auto
T07	NEC	v3.3.1	-	ON	Auto
T08	NEC	v3.3.1	-	OFF	Auto

Table 2: Test plan with ORCA2LIM_PISCES configuration

Test ID	Arch	NEMO ver.	SMT	Exclusive Mode	Dom. Decomp.
T09	IBM	v3.3	ON	ON	Manual
T10	IBM	v3.3.1	ON	ON	Manual
T11	NEC	v3.3	-	ON	Manual
T12	NEC	v3.3.1	-	ON	Manual

Table 3: Test plan with POMME configuration

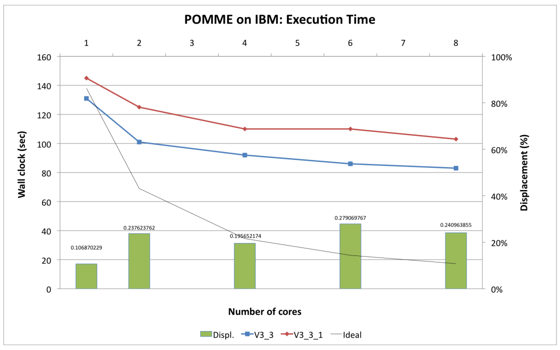
Test ID	Arch	NEMO ver.	SMT	Exclusive Mode	Dom. Decomp.
T13	IBM	v3.3	ON	ON	Manual
T14	IBM	v3.3.1	ON	ON	Manual
T15	NEC	v3.3	-	ON	Manual
T16	NEC	v3.3.1	-	ON	Manual

RESULTS AND COMMENTS

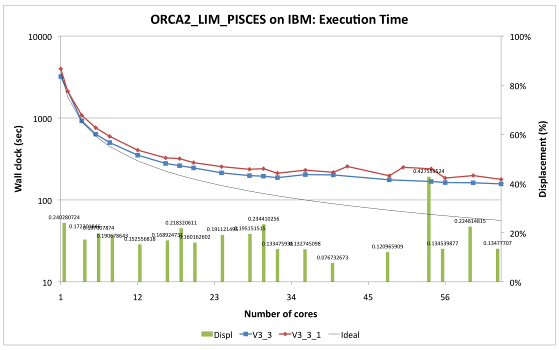
For each of the test described before we made a series of runs, considering the elapsed time and the memory allocation with a growing number of processes. In this section we analyze and describe the results.

EVALUATING THE PERFORMANCE OF THE NEMO V3.3 AGAINST NEMO V3.3.1

The comparison of both versions has been scheduled to be made on a IBM parallel machine based on Power6 processors and a NEC vector machine based on SX9 computing nodes. All the runs of the NEMO version 3.3.1 performed on the NEC architecture failed with a `floating-point zero divide error` in the *ldftra* module. This error does not allow us to make any kind of consideration about the computational performance comparison on the NEC architecture. For comparing the computational performances of the both versions of NEMO we took into consideration tests `T01`, `T02`, `T09`, `T10`, `T13`, `T14`. The Fig.1 and Fig.2 report the execution time and the measurement of the displacement in time between the NEMO version 3.3 and 3.3.1. In all of the analyzed configurations, the version 3.3.1 introduces a lose of performance. However, while for the POMME configuration the difference in time is meanly about 27%, it decreases to 15% for the ORCA2LIM_PISCES and up to 3.5% for MFS16. In general the more the configuration is computationally intensive, the more the overhead due to the dynamic memory management can be negligible. The Fig.3 and Fig.**??** report the evaluation of the maximum memory used for all of the considered configurations. For POMME and ORCA2LIM_PISCES configurations the total allocated memory of the NEMO version 3.3.1 is the same of the version 3.3 (the mean difference is less then 2%), while for MFS16 configuration we notice a more evident difference (about 5%) in terms of MBs of allocated memory by NEMO ver. 3.3.1. A deeper investigation is needed in order to justify this observations.
Taking into consideration only the MFS16 configuration we can evaluate the parallel speed-up and the efficiency of the parallel algorithm.

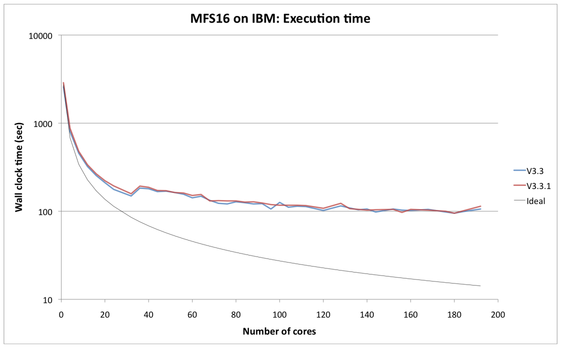


(a) POMME configuration

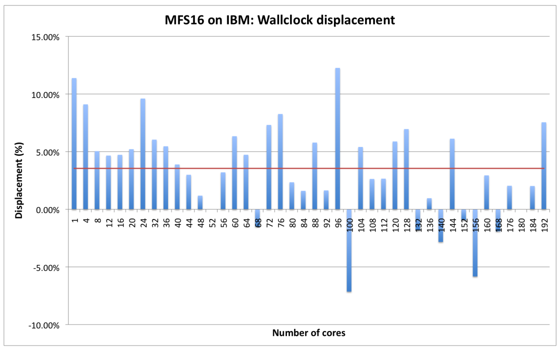


(b) ORCA2LIM_PISCES configuration

Figure 1: Performance comparison between NEMO ver. 3.3 and ver. 3.3.1

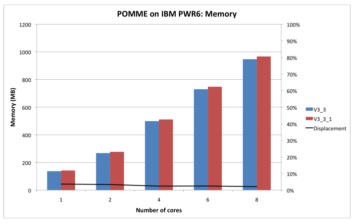


(a) MFS16 configuration

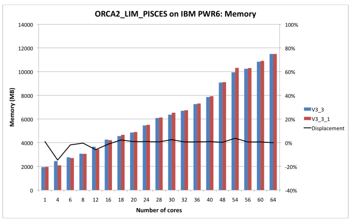


(b) Time displacement in MFS16 configuration

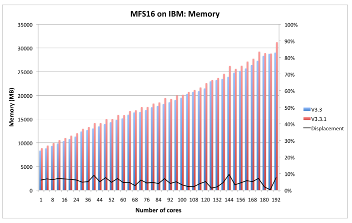
Figure 2: Performance comparison between NEMO ver. 3.3 and ver. 3.3.1



(a) POMME configuration

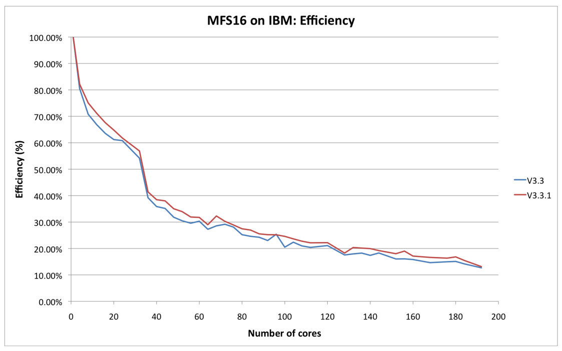


(b) ORCA2LIM_PISCES configuration

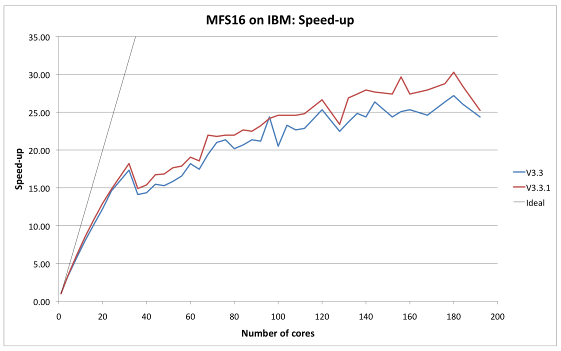


(c) MFS16 configuration

Figure 3: Comparison of the global memory used in NEMO ver. 3.3 and ver. 3.3.1



(a) Parallel efficiency



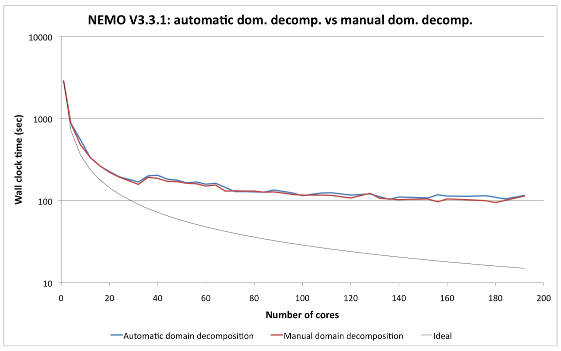
(b) Parallel speed-up

Figure 4: Comparison of the NEMO scalability between version 3.3 and 3.3.1 using MFS16 configuration

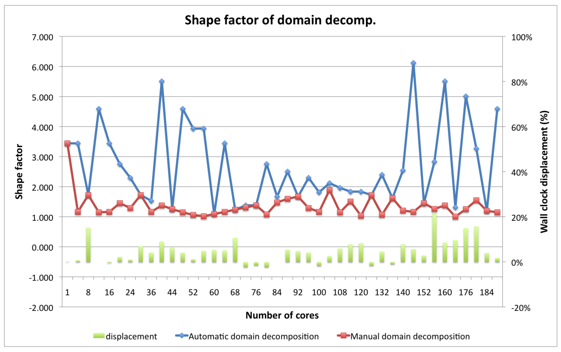
This configuration theoretically give us the possibility to scale the run with a higher number of processes rather than the other configurations. Fig.4 reports the results of this analysis. The model with this configuration scales well up to 32 processes, beyond this limit the performances start to decrease. It reaches an acceptable limit on 64 processes with an efficiency of 30%. Comparing the speed-up and efficiency from both versions, apparently the ver. 3.3.1 seams to perform better than ver. 3.3, but actually this is not true. The speed-up has been measured taking as reference the time of the parallel model launched on 1 process instead of the time of the best sequential algorithm. In this way, thus, we do not have the same sequential time as reference for both versions, and in particular the sequential time of the NEMO ver. 3.3.1 is greater then the sequential time of the NEMO ver. 3.3. We remind that the sole metric to be taken into account for comparing two or more models is the execution time. Fig.2a shows that the NEMO ver. 3.3 performs better then the NEMO ver. 3.3.1.

ESTIMATING THE EFFICIENCY OF THE DOMAIN DECOMPOSITION ALGORITHM

The dynamic memory allocation implemented in NEMO ver. 3.3.1 includes also the implementation of the algorithm for choosing the best domain decomposition with a given number of processes. The algorithm takes as input parameters the total number of processes, the dimension of the global domain and it returns the number of processes along latitude (`jpni`) and longitude (`jpnj`). Moreover, giving the bathimetry of the global domain, the algorithm can also establish the land sub-domains and eventually exclude them from the calculus. The domain decomposition plays a crucial role for the computational performance of the model; a good choice for domain decomposition will produce higher performance of the model.
Starting from the benchmark results described in [1] we can assert that the best domain decomposition generates the sub-domains with a shape as much squared as possible. This make minimum the overlap region among neighbor processes.
In order to evaluate the impact of the automatic



(a) Execution time with different domain decomposition strategies



(b) Shape factors for different domain decomposition strategies

Figure 5: Evaluation of the impacts of different domain decomposition strategies on the computational performances for NEMO ver. 3.3.1 on IBM cluster

domain decomposition we compared the results from test `T01` and `T05`. The Fig.5 reports the execution time of the NEMO ver. 3.3.1 when the domain decomposition is chosen manually in order to be the optimal one, and when the domain decomposition is chosen automatically from the algorithm. The results show that the automatic domain decomposition is not the best for all the runs. In particular using a manual decomposition following the rule of the shape factor as near as possible to 1, we can get a performance improvement on an average of 5%. The automatic domain decomposition should be revised considering that there are some cases where it introduces up to 20% of lost of performance. The MFS16 configuration has a global domain made of $871x253$ grid points; using 156 processes the automatic domain decomposition gives `jpni=39` and `jpnj=4` that implies sub-domains with a shape factor of 2.8; with 176 processes it gives even worst with `jpni=11` and `jpnj=16` that implies a shape factor of 5. A modified version of the *nemogcm.F90* has been released. The file includes the algorithm for establishing the best domain decomposition. The evaluation of this update will be performed in the near future.

EVALUATING THE IMPACT OF THE SMT ON IBM POWER6 CLUSTER

Simultaneous multithreading is the ability of a single physical processor to simultaneously dispatch instructions from more than one hardware thread context. Because in the Power6 processors there are two hardware threads per

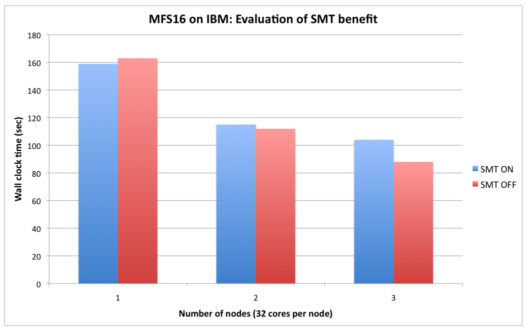
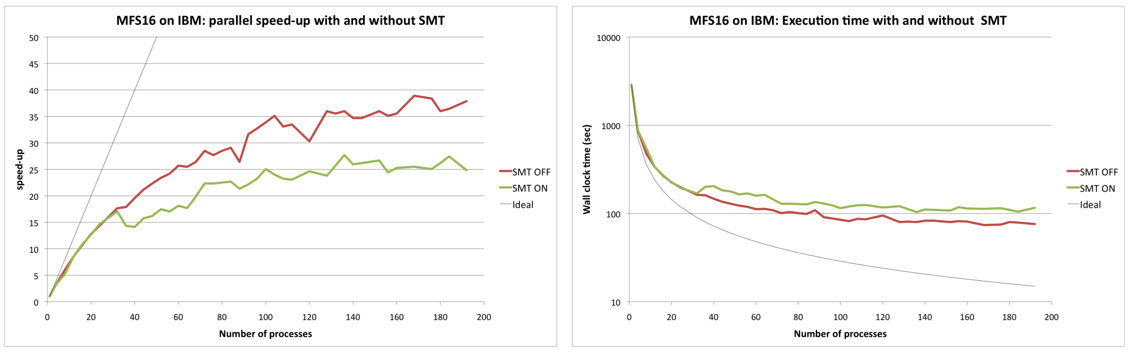


Figure 6: Evaluation of the benefit given by SMT for MFS16 configuration for NEMO ver. 3.3.1 on IBM cluster



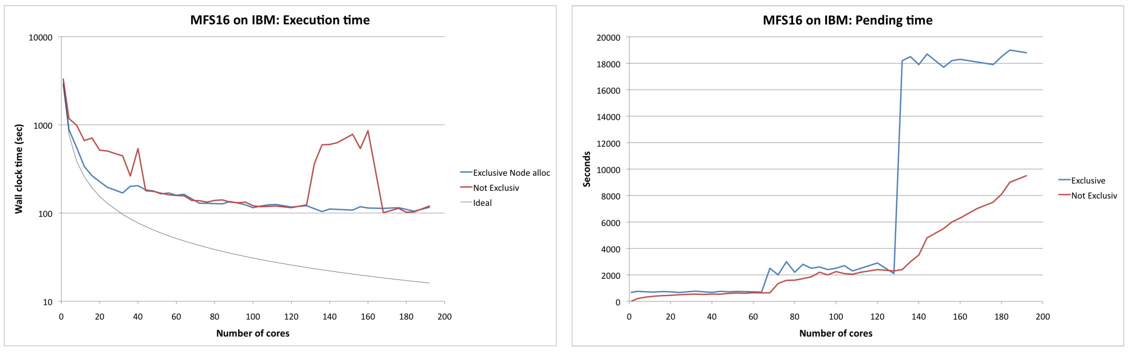
(a) Parallel speed-up (b) Execution time

Figure 7: Analysis of scalability for NEMO ver. 3.3.1 on IBM cluster, with and without SMT

physical processor, additional instructions can run at the same time. Simultaneous multithreading allows the user to take advantage of the superscalar nature of the processor by scheduling two threads at the same time on the same processor. Of course the advantage of the SMT is based on the idea that each single thread executed simultaneously does not fully saturate the resources of the physical processor. Applications that do not benefit much from simultaneous multithreading are those in which the majority of individual software threads use a large amount of any resource in the processor or memory. For example, applications that are floating-point intensive are likely to gain little from SMT and are the ones most likely to lose performance. These applications heavily use either the floating-point units or the memory bandwidth. Applications with low CPI (clock CycleS per Instruction) and low cache miss rates might see a some small benefit.

For these reasons an evaluation on the benefit of the SMT for the NEMO model is needed. The tests used to evaluate the SMT benefit are `T01` and `T03`. The Fig.6 reports the comparison of the execution time for the MFS16 configuration with and without SMT. In order to evaluate the SMT benefit we compared the execution time when the same computational resources are used and thus we took as reference the number of allocated nodes. When SMT is activated we used 64 processes to fully exploit a node; disabling the SMT a maximum of 32 processes per node can be allocated. The results show that the SMT does not provide any kind of benefit when the number of allocated nodes is greater than 1.

An analysis of scalability without SMT is reported in Fig.7. Although the number of processes is reported along the x-axis, different policies for process mapping are applied for both cases; when SMT is active, the local scheduler maps 2 processes for each physical core, while when the SMT is disabled only 1 process is mapped for each physical core. Having this in mind we can not use the charts in Fig.7 to compare or to evaluate the benefit of the SMT. Moreover when SMT is active, the scheduler distributes the MPI processes among the virtual cores using a stride=2. This mean that if the number of processes is less then 32 each process is mapped to a physical cores also in the case of SMT active.



(a) Execution time (b) Average pending time

Figure 8: Comparison of two cases: exclusive nodes allocation and shared nodes allocation using NEMO ver. 3.3.1 on IBM cluster

EVALUATING THE EXCLUSIVE ALLOCATION OF COMPUTATIONAL NODES AGAINST A SHARED NODES ALLOCATION

All of the previous tests have been performed using an exclusive node allocation, then even for a run with few processes a whole computing node has been allocated. This configuration can guarantee the best performance since the resources local to the node (memory, bus and internal switches) are not shared with other jobs. On the other hand this generally implies an inefficient use of the computing nodes for those cases when the number of processes/threads are less than the number of cores. The test presented in this section aimed at evaluating how much is the lost of performance if the computing nodes are not exclusively allocated for the NEMO simulations. It is worth noting here that the measures are deeply conditioned from the actual workload of the whole cluster. An exhaustive analysis would require the evaluation of the execution time for a long time period with different workload. Several runs have been executed during a week; in fig. 8a the execution time is reported.

As expected the wall clock time considerably increases with high values of processes, this is due to the spreading of the processes among several computing nodes. However it is interesting to evaluate also the mean pending time for each submission. In fig. 8b the pending time for jobs requiring exclusive node allocation is compared with ones that do not.

Bibliography

[1] I. Epicoco, S. Mocavero, and G. Aloisio. The nemo oceanic model: Computational performance analysis and optimization. *to be published in the HPCC2011 proceedings*, 2-4 Sep 2011, Banff Canada.

[2] P. Oddo, M. Adani, N. Pinardi, C. Fratianni, M. Tonani, and D. Pettenuzzo. A nested atlantic-mediterranean sea general circulation model for operational forecasting. *Ocean Sci.*, 5(4):461–473, 2009.