cmcc Centro Euro-Mediterraneo sui Cambiamenti Climatici

Research Papers
Issue RP0196
December 2013

Scientific Computing and Operations Division (SCO)

Optimal task mapping for NEMO model

By **Italo Epicoco**
University of Salento & CMCC
italo.epicoco@unisalento.it

Francesca Macchia
CMCC
francesca.macchia@cmcc.it

Silvia Mocavero
CMCC
silvia.mocavero@cmcc.it

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

SUMMARY The climate numerical models require a considerable amount of computing power. The modern parallel architectures provide the needed computing power to perform scientific simulations at acceptable resolutions. However, the efficiency of the exploitation of the parallel architectures by the climate models is often poor. Several factors influence the parallel efficiency such as the parallel overhead due to the communications among concurrent tasks, the memory contention among tasks on the same computing node, the load balancing and the tasks synchronization. The work here described aims at addressing two of the factors influencing the efficiency: the communications and the memory contention. The used approach is based on the optimal mapping of the tasks on the SMP nodes of a parallel cluster. The best mapping can heavily influence the time spent for communications between tasks belonging to the same node either to different nodes. Moreover, if we consider that each parallel task will allocate different amount of memory, the optimal tasks mapping can balance the total amount of main memory allocated on the same node and hence reduce the overall memory contention. The climate model taken into consideration is PELAGOS025 made by coupling the NEMO oceanic model with the BFM biogeochemical model. It has been used in a global configuration with a horizontal resolution of 0.25°. Three different mapping strategies have been implemented, analyzed and compared with the standard allocation performed by the local scheduler. The parallel architecture used for the evaluation is an IBM iDataPlex with Intel SandyBridge processors located at the CMCC's Supercomputing Center.

INTRODUCTION

The mapping is the process of assigning tasks to processors. In the case of multiprocessor architectures, the mapping is managed by a local scheduler, which decides how to assign the tasks to the compute nodes. The user can define a custom mapping too. However she must always keep in mind two aims conflicting each other:

- maximize the utilization of CPU
- minimize the communication among processes

We can recognize two mapping techniques:

- the static mapping, in which the tasks are mapped to the processes a priori (in this case it is necessary to know in advance the computational load of each task)
- the dynamic mapping, in which the tasks are mapped to the processes at runtime (useful, for example, when tasks are created during the execution or when the computational load of the task is unknown)

In this work, we have considered a static mapping to optimize the execution time of the PELAGOS025 model on the nodes of the Athena parallel cluster located at the CMCC Supercomputing Center. The PELAGOS025 is a coupled model among the NEMO oceanic model [13] and the BFM biogeochemical model [15] developed at CMCC. The model has been tested in a global configuration with a horizontal resolution of 0.25° and 50 vertical levels. It uses more than 50 pelagic variables. Three different partitioning strategies have been defined depending on the objective to be pursued. In particular:

- the minimization of inter-node communications
- the optimization of the memory access for each node
- the minimization of intra-node communications

These strategies have been compared to the default mapping defined by the scheduler, which distributes the tasks on the compute nodes according to the rank order.

MAPPING FOR MINIMIZING INTER-NODE COMMUNICATIONS

The main aim of this type of mapping is to minimize the communications among different computing nodes [4]. To accomplish this, the job is modeled by a graph, whose vertices represent the processes of the parallel program and edges the communication channels among them. This approach is well suited to irregular patterns of communication, which lacks a structured topology [5, 9]. In general, the purposes of partitioning are:

- to evenly distribute the vertices to nodes
- to minimize and balance the number of edges between nodes

The NEMO domain is represented by a regular grid and the computation in each subdomain requires its own data and those belonging to the neighbor subdomains according to the 5-points cross pattern. Each subdomain is also assigned to a task, associated with a processor. If the distribution of tasks takes place randomly on the nodes, it could have a large overhead due to communications. The optimal solution provides the partitioning of tasks into p parts so that:

- each part contains about the same amount of tasks (or vertices)
- the number of edges that cross the boundaries of a partition is minimal.

Moreover there are additional constraints due to the particular partitioning that we want to perform. Since the tasks are mapped on the nodes with a particular architecture, they must be organized in groups matching the number of cores on each node. Figure 2 shows an example of mapping that minimizes the inter-node communications. Finding the optimal partitioning is an NP-complete problem, but there are heuristics that approximate the optimal solution very well; in particular, in our case, the partitioning was calculated using the Scotch library [14]. As above mentioned, each task has been associated with an unweighted node on the graph and communications between the neighbors are modeled by arcs with the same weight. The algorithm used by the Scotch library minimizes the communications ensuring load balancing within a certain range; in our case, however, we imposed the constraint of mapping the same number of tasks on each node, so the tolerance on the load balancing has been forced to be zero. Denoting by S the graph, v_S the vertices and e_S the arcs, we have that $w_S(v_S)$ and $w_S(e_S)$ represent the computation weight of the corresponding process and the amount of data to be transmitted on the channel, respectively. The target machine when the parallel program runs is also modeled by an unoriented graph T. At the vertices v_T and edges e_T of T the integer weights $w_T(v_T)$ and $w_T(e_T)$ are assigned estimating the computational power of the corresponding processor and the cost of traversal of the inter-processor link, respectively. A mapping from S to T is obtained by minimizing the cost function obtained by weighing the cost of communications with the speeds of the link on which the communication takes place. Denoting by f_c the communication cost function and by $|\rho_{S,T}(e_S)|$ the number of edges of $E(T)$ used to route e_S, we have

$$f_C(\tau_{S,T}, \rho_{S,T}) = \sum_{e_S \in E(S)} w_S(e_S)|\rho_{S,T}(e_S)|$$

This function has several interesting properties: it is easy to be computed, allows incremental updates performed by iterative algorithms and its minimization favors the mapping of intensively intercommunicating processes onto nearby processors. The mapping algorithm implemented by the Scotch library uses a divide and conquer approach to recursively allocate subsets of processes to subsets of processors [2, 6, 7, 8]. It starts by considering a set of processors, also called domain, containing all the processors of the target machine, and with which the set of all the processes to map is associated. At each step, the algorithm bipartitions a domain not yet processed into two disjoint subdomains, and calls a graph bipartitioning algorithm to split the subset of processes associated with the domain across the two subdomains (Figure 1). Scotch expects as input a plain ASCII text file containing the graph S described by its adjacency matrix. The first line of a graph file holds the version number, (starting from version 4.0 is set to 0). The second

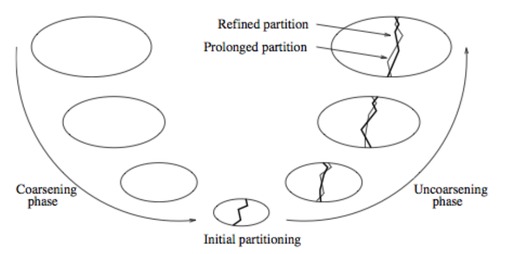

Figure 1:
Divide and conquer approach of the mapping algorithm implemented by the Scotch library.

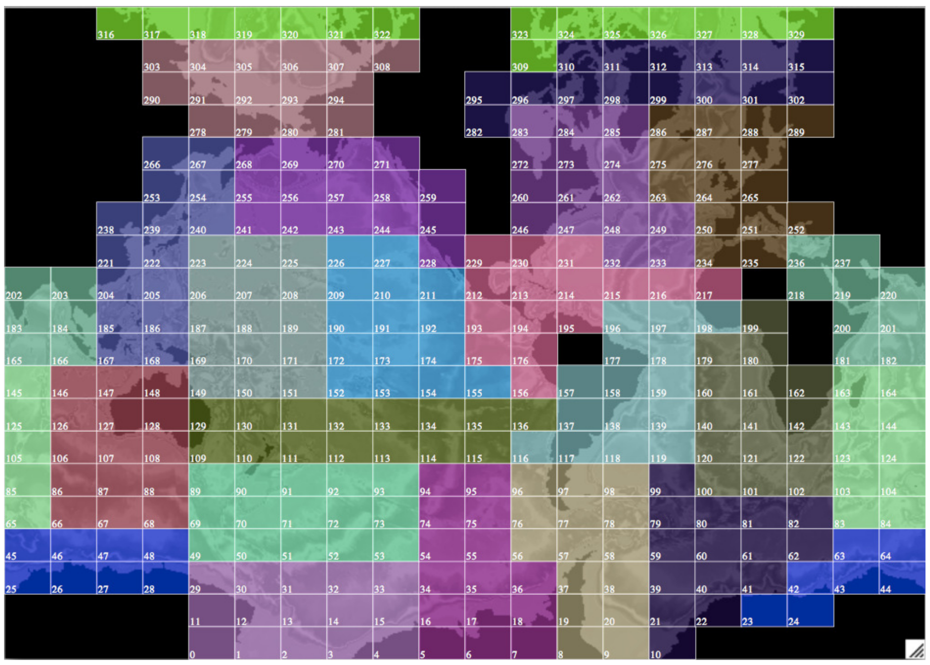

Figure 2:
Mapping for minimizing inter-node communications

line holds the number of vertices of the graph, followed by its number of arcs. The third line holds the graph base index value (baseval) and a numeric flag. The graph base index value records the value of the starting index used to describe the graph; it is usually 0 when the graph has been output by C programs and 1 for Fortran programs. Its purpose is to ease the manipulation of graphs within each of these two environments, while providing compatibility between them. The numeric flag is made of three binary digits and indicates if vertex or edge weights are provided or not. After these three lines of the header there are as many rows as the number of nodes in the graph. At the generic $i + 3$ row, we find all the nodes adjacent to the node i.

The input graph file for the Scotch library is obtained from the preprocessing tool named *cmcc_mppopt_showproc* (suitably modified) of NEMO; it generates a file containing a matrix that describes the domain. Each subdomain made of land points only is labeled with the value -1 and it is excluded by the computation, all of the other domains are labeled with a rank ID and associated with a node of the graph. As final step, a bash script is responsible to find, for each node of the graph, which ones are adjacent to it, following the communication pattern with 5-points stencil. Finally, it counts the number of arcs, writes the header of the file and completes the adjacency matrix of the graph. The file so created is used by the Scotch library, which will generate a new text file containing the partitions.

The output file is composed of:

- a header line containing the number of processed nodes

- one row for each node containing the node number followed by the partition number that is associated with

A final script will take care to read this file in order to obtain the mapping file to be passed to the scheduler. For our tests, the final script generates the task geometry string for the LSF scheduler.

MAPPING BASED ON MEMORY

In the mapping based on the optimization of the memory access the goal is to distribute tasks so that the total memory allocated on a single node is the same on all of the nodes. This way, it reduces the number of simultaneous accesses to the memory on the same node, thus reducing the time due to the memory contention. The mapping problem can be formalized through the resolution of a minimization problem. Denoting by M_i the memory allocated from the i-th process, with π_i the set of processes belonging to the i-th partition and P the total number of partitions, the mapping consists in defining all sets minimizing the following objective function:

$$f_C = max_p \left\{ \sum_{i \in \pi_p} (M_i) \quad \forall \ 1 \leq p \leq P \right\}$$

As we have seen for mapping based on communication, there is the constraint that all partitions must have the same cardinality (perfect balancing). To solve the exposed problem of minimum we have used a heuristic algorithm derived from the Max-Min scheduling algorithm. Denoting by $M = \{M_0, M_1, M_2, \ldots, M_t\}$ the set of values of the memory required by tasks that are not assigned to any partition, π the set of all partitions and C_i the cost associated with each partition calculated as the sum of the memory of all tasks associated with the i-th partition, we proceed as described in the following pseudocode:

```
foreach i {
   Ci <- 0
}

foreach m in M {
   let Mi = max{M}
   let Cp = min{C}
   push i into PI{p}
   Cp <- Cp + Mi
   remove Mi from M
}
```

MAPPING FOR MINIMIZING INTRA-NODE COMMUNICATIONS

The third mapping strategy used is based on the minimization of intra-node communications, so it tries to minimize the number of messages exchanged between tasks mapped on the same node. This strategy is exactly contrary to the former one, which was based on the minimization of communications among different nodes, and then it maximizes intra-node communications. The main motivation for the choice of this strategy comes from some results available in literature [3, 10, 11]. In the SGI Altix architecture, the cross pattern communication is more efficiently if achieved through the distribution of near tasks on remote nodes. From the algorithmic point of view, the mapping is done by considering that each process borders at most with the four processes close to it. The global domain is divided on a two-dimensional grid, so each process can be uniquely identified by a row index i and a column index j. In the current mapping strategy, also called neighbor-away, if the process (i, j) is assigned to the partition p, then the processes $(i + 1, j)$, $(i - 1, j)$, $(i, j + 1)$ e $(i, j - 1)$ will be allocated on a partition different from p (as we can see in the figure 3). This mapping strategy always allows canceling the intra-node communications when the number of partitions is greater than

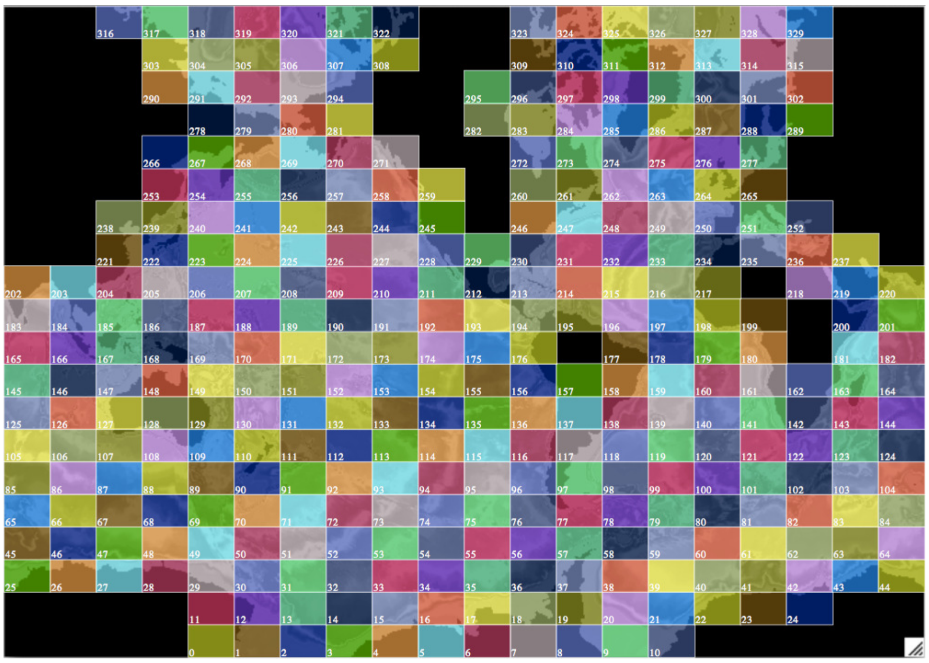

Figure 3:
Mapping for minimizing intra-node communications

or equal to 2. With two partitions, for example, the optimal partitioning is obtained by splitting the processes as on a chessboard; assigning the black cells to the first partition and the white cells to the second one, we could avoid communications among processes belonging to the same partition.

DEFINITION OF THE ANALYTICAL MODELS

The mapping strategies described in the previous sections can be applied if we know the data related to the communications cost or the memory allocated by each process. In this section we present the analytical models for the estimation of the memory required by each process and the communication model that estimates the time to transfer data between a process and its adjacents.

MEMORY MODEL

The parallelization used in NEMO model is based on a domain decomposition; each process has to elaborates a subdomain of dimension jpi x jpj x jpk including the overlap region between the process and its neighbors. The subdomain size is inversely proportional to the number of processes. The data structure used in NEMO is based on tridimensional matrix with $jpj * jpi * jpk$ floating-point elements for each physical variable. BFM instead uses a data structure with mono-dimensional arrays in which the values of the pelagic variables are stored only for the ocean points; so, in general, the number of elements in a pelagic array is smaller than the dimension of the subdomain. PELAGOS025 is a coupled model, so we have to consider the data structure of both NEMO and BFM. In general the memory allocated by each process is given by a term directly proportional to the subdomain dimension (according to the data allocated in NEMO), directly proportional to the number of ocean points in the subdomain (according to the data allocated in BFM) and a constant quantity of memory related to scalar variables and data needed for parallel processes management. The memory model can be formalized by the equation

$$M = \alpha \cdot jpi \cdot jpj \cdot jpk + \beta \cdot Opt + \gamma$$

where jpi, jpj and jpk represent the dimension of the subdomain along the three directions and Opt is the number of ocean points in the subdomain. As in a linear model we can evaluate the coefficients α, β and γ using a linear regression. The configuration used

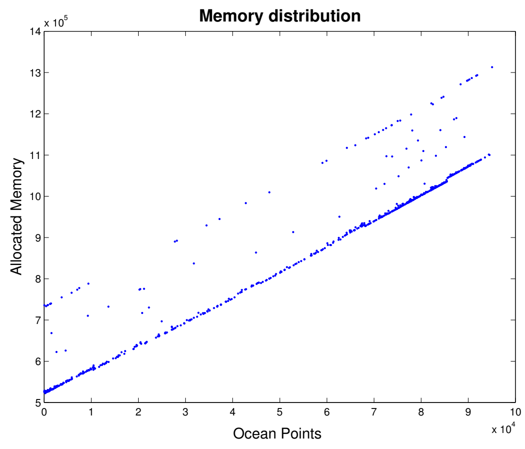

Figure 4:
Memory distribution related to the number of ocean points given by a 19x45 domain decomposition.

to evaluate the coefficients is based on a simulation executed on 672 processes; for each process we have measured the total amount of allocated memory. The number of ocean points of each subdomain is evaluated using the input file of the model named bathymetry storing the pelagic depth in each point of the global domain. Figure 4 shows the memory evaluated for the configuration with 672 processes. On the y-axis the quantity of memory located by the subdomain containing the number of ocean points reported on the x-axis, is shown. Table 1

Coefficient	Value
α	1.030 KB
β	6.142 KB
γ	421.44 MB
R^2	97.49%
Standard error	62.62 MB

Table 1
Estimate of the memory model coefficients given by linear regression. α represents the memory allocated for each point in the domain, β is the memory allocated for each ocean point and γ stands for the memory needed for scalar variables

reports the evaluation of the coefficients obtained with the linear regression, the standard error and the coefficient of determination R^2, that evaluate the differences between the values of memory estimated and measured. This coefficient can assume values between 0 and 1. A value of 1 means that there is a perfect correlation, i.e. there is no difference between the estimated value and the actual one. At a later stage the model is checked out on other decompositions. Table 2 reports the value of the root mean square error (RMSE), expressed in GigaBytes, for each examined decompositions. This value represents the square root of the mean-square error. The relative RMSE, instead, expresses the root mean square error compared to the average of the examined sample. The relative RMSE is always less than 6%, so we can assume that the memory model estimates with a good approximation the actual trend. This model can be refined analyzing more in detail the observed data. In fact for some domains an amount of constant memory independent from the number of ocean points, but related to the size of the domain, is allocated.

Decomp.	Relative RMSE	RMSE (GB)
160	4.651%	0.0995
192	5.106%	0.0950
224	4.647%	0.0763
256	5.489%	0.0813
288	5.773%	0.0790
320	5.568%	0.0698
512	4.907%	0.0473

Table 2
Root mean square error (in absolute value and relative to the mean) measured for some decompositions.

COMMUNICATION MODEL

The communications among processes in the NEMO model are based on a cross communication pattern with a 5-points stencil, that is each process sends data to the four neighbors. The communications are implemented using point-to-point MPI calls. To ensure the exchange of the elements at the corners, there are two communication phases: in the first phase all processes only exchange in the east-west direction, then swap along the north-south direction. In each of the two phases the generic process implements a nonblocking send (MPI_Isend) to the neighbors, then calls a blocking receive to obtain the corresponding values and finally invokes the MPI_Wait to check for the send completion. Based on this implementation we can define a communication model envisioning a crossbar interconnection between computing nodes of the parallel cluster. In the communication model we have assumed a full duplex channel, i.e. the channel can be used simultaneously to receive and to send data to the same recipient. The communication time will be directly proportional to the amount of exchanged data plus a latency time to establish the connection. Considering that during the exchange in the east-west direction each process must send the entire face formed by $jpj * jpk$ elements, while during the exchange in the north-south direction the amount of data is equal to $jpi * jpk$, indicating with T_w the time required to transfer an element and with T_s the startup time of the communication, we can model the communication time in the following way:

$$T_{com} = 2 * (jpi + jpj) * jpk * T_w + 4T_s$$

T_w and T_s depend on the type of interconnection: if the communicating processes are on the same computing node, the exchange of messages takes place through shared memory without involving the interconnection network; otherwise, they depend on the transmission speed of the interconnection network. For the evaluation of the parameters T_w and T_s we used the benchmark included in the suite HPCC (High Performance Computing Challenge) [12, 1], which is widely recognized in the context of high performance computing. In particular we used the *b_eff* benchmark, which measures the latency and the bandwidth of the interconnection system. Table 3 shows the latency and bandwidth for the cluster Athena. The communication time is smaller

	Inter-node	**Intra-node**
Bandwidth (GB/s)	0.2	1.1
Latency (μ sec)	5.5	0.9

Table 3
Bandwidth and latency for inter- and intra-node communications referred to the Athena cluster and measured with the b_eff benchmark

when the communications occur between processes mapped on the same node. However, the proposed model is difficult to apply since we have to identify for each communication if it occurs inter- or intra-node.

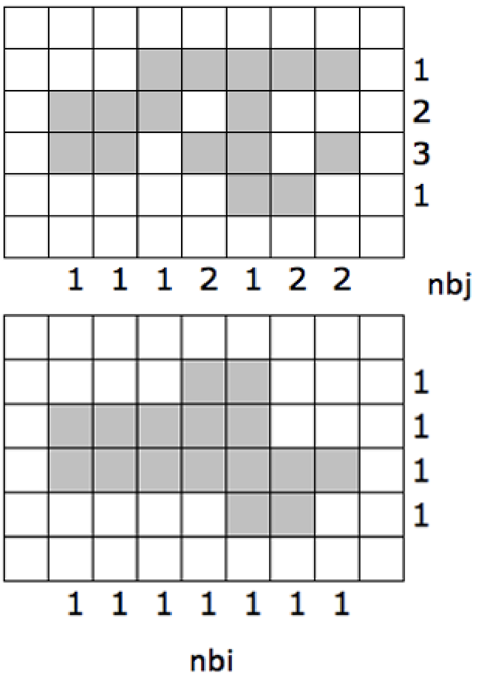

Figure 5:
Example of processes allocation on a compute node. Gray cells represent processes allocated on the same node.

The inter-node communications can be formalized topologically describing the area of the processes assigned to the same node and using two sets, *nbi* and *nbj*.

The element nbi_j represents the number of blocks of contiguous processes along the column j, while the value nbj_i indicates the number of blocks of contiguous processes along the row i.

Looking at the figure 5 and imagining that the processes in gray are assigned to the same node, the values of nbi and nbj are listed at the end of each row and each column. The number of inter-node communications will then be equal to

$$N_{com} = 2(\sum nbi_j + \sum nbj_i)$$

The amount of data exchanged with another node will be equal to

$$Msg = 2 * jpk * (jpi * \sum nbi_j + jpj * \sum nbj_i)$$

Finally, the inter-node communication pattern can be expressed by

$$T_{internode} = Msg * T_w + N_{com} T_s$$

Let's consider separately the two components that contribute to the communication time: the latency, related to the communication start time, and the bandwidth, related to the data actually transferred.

To evaluate the data transfer component, we consider the region as a whole domain allocated on the same node. This area will have a number of points equal to $jpi * jpj * jpk * np$ (where np is the number of processes per node). Taking the number of processes per node constant and decomposing only along the i and j directions, the transfer time is minimal if the perimeter of the above mentioned area is minimal. Maintaining a constant area, the perimeter is minimal in the case of square area and that is when

$$jpj * \sum nbi_j = jpj * \sum nbj_i$$

$$\sum nbi_j * \sum nbj_i = np$$

Instead, to minimize the latency, we have to minimize the number of communications among different nodes, regardless of the size of the message. The minimum is obtained again by placing these processes on a square and then

$$\sum nbi_j = \sum nbj_i = \sqrt{np}$$

Finally, we have to notice that the described model takes into account only the communication time and does not consider any load imbalance among processes. The load imbalance generates an idle time of the faster processes waiting for the most computationally burdened process.

PERFORMANCE ANALYSIS

A set of tests were carried out to verify the scalability of the model using the above described mapping strategies. This way it was possible to evaluate the impact of different strategies and to estimate the improvement they bring.

COMPUTING ENVIRONMENT

The tests were carried out on the Athena cluster, located within the CMCC Supercomputing Center. Athena is a parallel cluster with 482 IBM iDataPlex dx360 M4 nodes, each one composed of:

- two Intel Xeon E5-2670 SandyBridge 2.6 GHz and 8 cores, for a total of 16 cores per node;
- 64GB of RAM (4 GB for each core);
- three cache levels, 32KB, 256KB and 20MB respectively.

The first two cache levels are local to each core, and the last one is shared by all the 8 cores inside the processor. Thanks to the vector unit and the AVX instruction set, each core is able to perform 8 double precision floating point operations per clock cycle, expressing a calculating capacity for each node equal to

$$16\ cores/node\ *\ 8\ FLOP/core\ *\ 2,6\ GHZ$$

$$=\ 332.8\ GFlops/node$$

The entire cluster is able to perform 160.41 TFlops. In addition, the processor supports a 2-way Simultaneous Multi Threading (SMT), which allows to run two threads simultaneously on each core; so, activating SMT, we can run 32 threads per node. The nodes are interconnected by two networks: the first one is a Gigabit Ethernet and is used for the cluster management; the second one is a Mellanox InfiniBand FDR 56Gb/s, which is used to exchange messages during the execution of MPI parallel applications. The parallel environment includes MPI libraries, numerical libraries and compilers developed by IBM and optimized for the architecture. The allocation of computing resources to the jobs is handled by the LSF scheduler.

TEST PLAN

The model PELAGOS025 taken into account is characterized by a global domain size of 1442 x 1021 x 50 grid points. This domain has been divided into $jpni$ x $jpnj$ subdomains and, eliminated those containing exclusively land-points, each of the remaining subdomain has been assigned to a processor. The greatest decomposition tested on Athena contains 7680 subdomains partitioned as 155 x 73. Starting from it, several configurations adequately spaced, were defined in order to experimentally define the scalability curve. The tests were planned to be performed on a number of processors ranging from 160 to 7680. Unfortunately, when the number of processors exceeds 3488, the application failed due to a bug at run time.

The execution was then reformulated in seven decompositions, summarized in more detail in table 4. The execution of each run is preceded by the execution of a script, that allows to define the parameters for setting up the machine; it analyzes an input parameter that specifies which of the four types of mapping we want to use and takes the value

Decomposition	Number of processors
6 x 29	160
38 x 18	544
52 x 24	944
104 x 17	1344
70 x 34	1728
122 x 23	2048
63 x 79	3488

Table 4
List of the decompositions tested on Athena

card in which the processes are mapped to partitions according to the rank number;

away which minimizes the intra-node communications;

mem which allows to obtain a mapping based on the memory;

comm which minimizes the inter-node communications.

Each of these types of mapping was tested on each decomposition indicated in the table 4. Each test was repeated three times in order to mitigate the effect of jitter introduced by the operating system and the management services of the cluster, for a total of 84 executions. In addition, each run was performed at empty machine exclusively dedicated to the running job.

RESULTS

In this section we summarize the results obtained testing the reference decomposition on Athena. Each run carried out a simulation of 40 time steps. The execution time for a single time step and the average on all of the time steps have been measured. The time of the first, the twentieth and the last two time steps have been excluded from the analysis, since they perform I/O operations. In table 5 we report the data for the wall-clock time estimated only for the com-

	card	*away*	*mem*	*comm*
160	25.04	22.06	22.94	25.02
544	6.69	6.28	6.10	6.73
944	3.89	3.43	3.55	3.90
1344	3.04	2.63	2.62	3.30
1728	2.24	1.86	1.90	2.23
2048	2.02	1.61	1.73	2.02
3488	1.39	1.27	1.25	1.28

Table 5
Wallclock time of PELAGOS025 model on Athena with different mapping strategies

putation. Figure 6 shows as the computational time for each time step decreases significantly using the *away* and *mem* strategies compared with the mapping used by the scheduler (*card*), while the computation time for each step remains almost unchanged using the *comm* strategy.

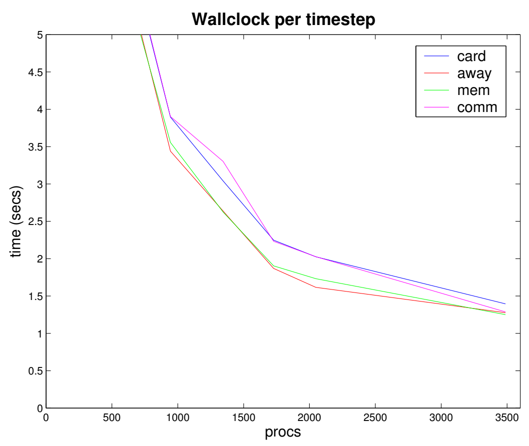

Figure 6:
Wall-clock time of PELAGOS025 model on Athena with different mapping strategies

This means that a mapping strategy balancing the memory allocated on each node performs better than one that tries to minimize inter-node communications.

This result can be justified by the type of architecture. As previously described, the computing nodes of Athena are interconnected via a Mellanox InfiniBand FDR 56Gb/s network, therefore the exchange of MPI messages during the execution of parallel applications is very fast; decreasing the number of these communications do not therefore leads to considerable benefits.

Different considerations, however, can be made for the *mem* strategy; the distribution of the tasks so that the total memory allocated on a single node is the same on all of the nodes, reduces the number of simultaneous accesses to the memory on the same node, reducing also the time for the memory contention.

Some other improvements are due to the use of the *away* mapping strategy. As above described, it aims at minimizing the number of messages exchanged between tasks mapped on the same node.

The success of this mapping strategy depends on the configuration of the problem.

The subdomains including coastal areas also include land-points, which are not involved in the computation; so these subdomains will require a memory load lower than those that include only ocean points.

Moreover, our model takes into account not only the geographical area of the partition, but also the depth of the water at those points; so the subdomains that require more memory will be concentrated in the deepest parts of the ocean. Thus, we can deduce that closed subdomains require approximately the same amount of memory; a mapping strategy placing the closed processes on different nodes will also produce a memory balancing, returning in the *mem* strategy case.

These considerations are also supported by measuring the memory balance among nodes, as shown in the table 6.

In order to highlight the improvements made by the use of different mapping strategies (*away*, *mem* and *comm*), table 7 reports the percentage of improvement that they bring compared to the mapping used by the scheduler (*card*).

Figures 7 and 8 show respectively the parallel speedup and efficiency for the different mapping strategies.

Since each time step simulates a time interval of 1080 seconds we can estimate the number

	card	*away*	*mem*	*comm*
160	1.56%	1.09%	1.00%	1.56%
	124	610	564	124
544	1.44%	1.19%	1.00%	1.42%
	1118	2084	2040	522
944	1.38%	1.18%	1.00%	1.46%
	1942	3614	3568	916
1344	1.45%	1.18%	1.00%	1.43%
	2734	5184	5130	1306
1728	1.37%	1.19%	1.00%	1.39%
	3512	6632	6568	1728
2048	1.37%	1.09%	1.00%	1.34%
	4126	7886	7842	1974
3488	1.27%	1.16%	1.00%	1.30%
	7286	13454	13416	3590

Table 6
Memory balancing expressed as percentage between the most loaded node over the average one and the inter-node communication links evaluated for different decompositions and different mapping strategies

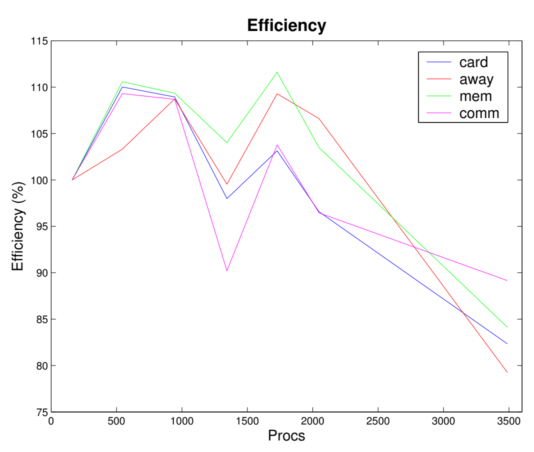

Figure 8:
Parallel efficiency of PELAGOS025 model on Athena with different mapping strategies

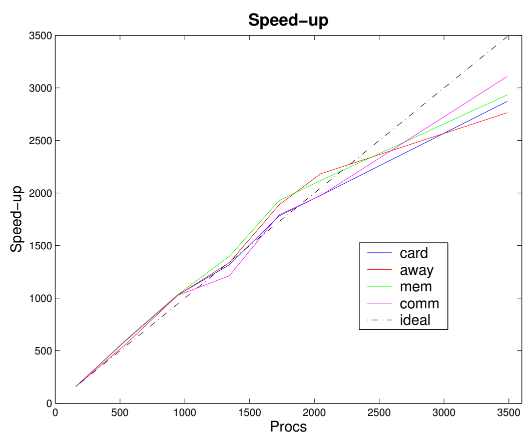

Figure 7:
Parallel speedup of PELAGOS025 model on Athena with different mapping strategies

	away	*mem*	*comm*
160	11.89%	8.37%	0.05%
544	6.18%	8.84%	-0.61%
944	11.72%	8.72%	-0.19%
1344	13.26%	13.69%	-8.56%
1728	16.85%	15.32%	0.66%
2048	20.19%	14.50%	-0.07%
3488	8.45%	10.31%	7.69%

Table 7
Speedup for each mapping strategy referred to the default one (*card*) defined by the local scheduler

of Simulated Years Per Day (SYPD) that represents the amount of years that we can simulate in 24 hour of run (Figure 9).

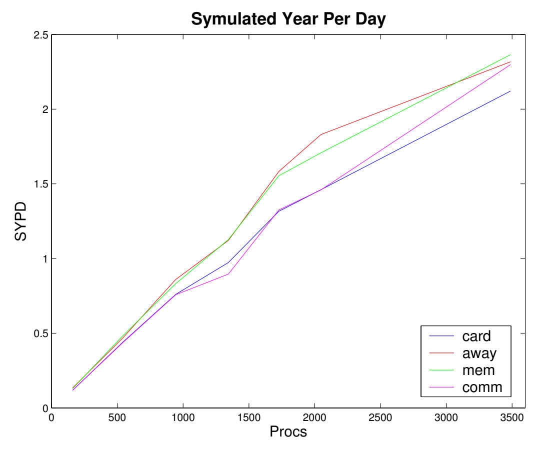

Figure 9:
Amount of years that can be simulated in 24 hours of calculation

CONCLUSION

The tests carried out show that a mapping strategy that balances the memory allocated on each node is better than one that aims to minimize inter-node communications. The optimization of the use and access to the memory plays an increasingly important role in exascale perspective. In fact, the new architecture will be designed by increasing the number of cores inside the cluster, but this will not be followed by an equally rapid increase in the amount of RAM memory. This implies that the amount of memory for the single core will decrease, making the strategies of optimizations that aim to reduce access to the memory more useful than others. Another important consideration is emerged in the analysis of the mapping strategy that minimizes the intra-node communications; due to the particular configuration of the problem, *away* mapping implies a balancing of the memory and, consequently, an improvement almost equivalent to *mem* type. Finally, due to the low latency and high bandwidth, a type of mapping based on the minimization of inter-node communications is almost completely irrelevant in the problem optimization. More in general, the strategy of the mapping is successful, as it involves an average improvement of 10% in execution time. This is a very useful result, whereas it has not been modified the source code.

Bibliography

[1] Hpcc website - http://icl.cs.utk.edu/hpcc/.

[2] S. T. Barnard and H. D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101–117, 1994.

[3] J. Donners, N. Audiffren, J.-M. Molines, A. Lecointre, and A. Coward. Towards petascaling of the nemo ocean model. In *Proceeding of EGU conference*, Vienna, April 23-27th 2012.

[4] F. Ercal, J. Ramanujam, and P. Sadayappan. Task allocation onto a hypercube by recursive mincut bipartitionning. *Journal of Parallel and Distributed Computing*, 10:35–44, 1990.

[5] S. W. Hammond. *Mapping unstructured grid computations to massively parallel computers*. PhD thesis, Rensselaer Polytechnic Institute, Troy, New-York, February 1992.

[6] B. Hendrickson and R. Leland. Multidimensional spectral load balancing. Technical Report SAND93-0074, Sandia National Laboratories, January 1993.

[7] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301, Sandia National Laboratories, June 1993.

[8] B. Hendrickson, R. Leland, and R. Van Driessche. Skewed graph partitioning. 8th SIAM Conference on Parallel Processing for Scientific Computing, IEEE, March 1997.

[9] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. Technical Report 95-064, University of Minnesota, August 1995.

[10] A. Lecointre, N. Audiffren, and J.-M. Molines. Calcul massivement parallèle dans un contexte de modélisation océanique. Technical report, LEGI-CNRS, Grenoble, March 28th 2012.

[11] A. Lecointre and J.-M. Molines. Tests de scalabilité sur orca025.l75 et orca12.l46 avec jade2. Technical report, LEGI-CNRS, Grenoble, France, 2011.

[12] P. Luszczek and D. Koester. Hpc challenge v1.x benchmark suite. In *Proceeding of SC05 conference*, Seattle, Washington, November 13 2005.

[13] Gurvan Madec and the NEMO System Team. Nemo ocean engine. Technical report, Pole de modelisation de l'Institut Pierre-Simon Laplace, Jan 2011.

[14] F. Pellegrini and J. Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *Proceeding of HPCN'96*, volume 1067 of *LNCS*, pages 493–498, Brussels, April 1996.

[15] The BFM System Team. *The Biogeochemical Flux Model (BFM) Equation Description and User Manual BFM version 5 (BFM-V5)*.