Original Article

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
1–18
© The Author(s) 2017
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/1094342016684930
journals.sagepub.com/home/hpc
SAGE

Hybridisation strategies and data structures for the NEMO ocean model

Italo Epicoco[1,2], Silvia Mocavero[2], Andrew R Porter[3], Stephen M Pickles[3], Mike Ashworth[3] and Giovanni Aloisio[1,2]

Abstract
This work describes the introduction of a second level of parallelism based on the OpenMP shared memory paradigm to NEMO, one of the most widely used ocean models in the European climate community. Although the existing parallelisation scheme in NEMO, based on the MPI paradigm, has served it well for many years, it is becoming unsuited to current high-performance computing architectures due to their increasing tendency to have fat nodes containing tens of compute cores. Three different parallel approaches for introducing OpenMP are presented, discussed and compared on several platforms. Finally we have also considered the effect on performance of the data layout employed in NEMO.

Keywords
shared-memory parallel paradigm, MPI–OpenMP, hybrid parallelisation, performance evaluation, NEMO ocean model

1 Introduction

NEMO (Nucleus of the European Model of the Ocean) (Madec and the NEMO Team, 2008) is one of the most widely used ocean models, of great significance to the European climate community. The NEMO model is written in Fortran90 and its origins can be traced back over 20 years and, as such, the code base, parallelised using the message passing paradigm, has seen many computer architectures and programming environments come and go. The model is used in more than 240 projects in 27 countries for climate studies both at regional and global scales. Portability, longevity and computational performance are therefore highly important to the NEMO users and developers.

The next generation of computer architectures will enable the resolution and the complexity of climate models to be increased, exploiting the resources' capability. However, for this to be fully realised, the scalability of current codes has to be improved, e.g. through the design of new parallel models. Indeed, when the number of parallel processes increases up to thousands of cores, the main bottlenecks to scalability are the communications overhead and the accessing of memory. More generally, data *movement* at all levels becomes the main limiting factor. The introduction of a hybrid programming model, based on message passing combined with a shared memory paradigm, enables a reduction in inter-process communication, an improvement in data locality (due to the sharing of data cache between cores) and can improve the load balance due to a more dynamic allocation of tasks/data to the available resources. Hybrid Message Passing Interface MPI-OpenMP parallelisation strategies have become increasingly important with the commodification of multi-core, shared-memory nodes (see, e.g., Smith and Bull, 2001)). Some examples are the nodes of Cray's XE6 and IBM's BlueGene/Q machines which have hardware support for 32 and 64 threads, respectively. Increasingly, high-end machines are being equipped with accelerators such as GPUs or Intel's Xeon Phi. These devices have many lightweight cores and writing efficient software for them can be complex.

The NEMO model is parallelised using the MPI with a two-dimensional geographic domain decomposition. The initial domain described along the latitude, the longitude and the depth coordinates is spread to the parallel tasks dividing the horizontal grid in a regular mesh. Each task takes only one sub-domain made of a

[1]University of Salento, Italy
[2]Euro-Mediterranean Center on Climate Change Foundation, Italy
[3]Science & Technology Facilities Council, UK

Corresponding author:
Italo Epicoco, Department of Engineering for Innovation, University of Salento, via per monteroni, 73100 Lecce, Italy.
Email: italo.epicoco@unisalento.it

rectangular portion of the horizontal grid and all of the corresponding vertical levels (i.e. the sub-domain is a parallelepiped). Each sub-domain is enriched with a 1-point wide halo region that is used to exchange the data at the borders with the neighbouring tasks. The communication pattern is based on a 5-point stencil and uses blocking MPI communications.

This parallelisation approach is common to many other ocean and fluid dynamics models since it is a natural way to exploit the fact that the same operations are performed at each point on the simulation grid.

The POP2 model (Parallel Ocean Program) Smith et al. (2010) is the ocean model developed by Los Alamos National Laboratory (LANL) in the United States and it provides the ocean component of the CESM (Community Earth System Model). POP2 exploits the available parallelism through domain decomposition, dividing the gridded domain along the longitudinal and latitudinal dimensions. Multiple blocks can be distributed to a single processing element and this provides more flexibility and a natural mechanism for hybrid parallelisation in which the threaded parallelism (e.g. OpenMP) can be used to distribute work within a multi-processor shared memory node, while message passing (e.g. MPI) is still used to communicate between distributed nodes. The use of small blocks can eliminate more land-only blocks, provide better load balancing and better performance on cache-based microprocessors. However, because each block has a halo region, blocks that are too small have a high surface-to-volume ratio and communication and synchronisation issues can become dominant.

The Weather Research and Forecast (WRF) model (Michalakes et al., 2005) was developed as a collaborative project by various organisations in the United States. It is a regional- to global-scale model intended for both research applications and operational weather-forecast systems. WRF uses a finite difference scheme on a regular, longitude–latitude mesh and can be built as a serial, distributed-memory or hybrid executable. Distributed-memory parallelism is implemented by decomposing the simulation domain into a regular grid of patches. Each MPI process is then assigned one patch. Each rectangular patch is then further decomposed into a regular grid of tiles (configurable at run-time). The number and shape of these tiles may be tuned for efficient cache usage and, for hybrid parallelisation, these tiles are shared amongst OpenMP threads. This approach has been shown to perform well on previous generations of supercomputer (Porter and Ashworth, 2010).

The Integrated Forecast System (IFS) (ECMWF, 2015) is an operational global meteorological forecasting model developed and maintained by the European Centre for Medium-Range Weather Forecasting (ECMWF). In contrast to POP2, WRF and NEMO it uses a spectral method rather than a finite difference scheme. The parallelisation of such a model has to not only deal with the data structure for 3D grid-point model data, but also with Fourier and spectral fields. IFS takes into account three levels of parallelisation; inter-node parallelisation (through the message passing paradigm), intra-node parallelisation (through the shared memory paradigm) and CPU (instruction)-level parallelisation exploiting the processor's vector unit. The grid point dynamics and physics computations contain only vertical dependencies, hence a domain decomposition based on longitudinal and latitudinal directions is the optimal choice. The partitioning algorithm used in IFS is based on the work of Paul Leopardi (Leopardi, 2006) and it is characterised by an increasing number of partitions in bands from the poles to the equator. The number of bands and the number of partitions in each particular band are derived so as to provide partitions of equal area.

Our work aims at enhancing the computational performance of NEMO through the use of a shared memory parallel approach merged with the distributed memory parallel approach already implemented in the model. We also experiment with different data partition algorithms to better exploit both the distributed memory and shared memory environment. Our main motivation in undertaking this work is that the parallel architectures of the near future will still be based on a cluster of nodes but with each node equipped with both accelerators and many-core processors making it much more important to leverage a shared-memory environment. In this work we have critically investigated the performance of various approaches for the introduction of OpenMP parallelism into the NEMO code (i.e. moving to a hybrid programming model) and we have also considered the effect on performance of the data layout employed in NEMO.

Section 2 describes the strategies and the approaches proposed for introducing thread-based parallelism. Since the introduction of OpenMP parallelisation implies also a reshaping of the original arrays and the way each element in an array is accessed, it is possible that a re-ordering of the array indices would bring benefits; Section 3 describes the impact on performance of the array index reordering. Following that, we give the definition of the test cases and the analysis of the results that we have obtained on seven different architectures. The conclusions and final remarks are given in the last section.

2 Hybridisation strategies

The NEMO model solves the primitive equations describing the state of the ocean using a finite-difference scheme on a simply connected, latitude–longitude grid. The grid is tri-polar and rotated such that the poles fall over the land and therefore do not cause problems for the numerical solution of the equations. The NEMO model operates on state variables defined on the three dimensions (indexed with i, j and k), hence the code is mainly characterised by triply nested loops. Each array is indexed with i (longitude) as the first index and k (depth) as the last, hence in the triply nested loops the outer loop is over k, then there is the loop over j and the inner loop is over i. Each MPI task exchanges, with its neighbours, the elements on the surface of its sub-domain; the MPI message size is then proportional to the horizontal perimeter of the block by the number of vertical levels.

In this section, three different approaches for introducing OpenMP to the NEMO code will be described. Each of these approaches has been evaluated taking into account the efficiency of the solution (in terms of time to solution and parallel speedup), and the software engineering aspects. Namely, the main criteria used for evaluating each approach have been:

- the *level parallelism*, i.e. the granularity of each parallel task;
- the *degree of load balancing*;
- the *introduction of performance penalties*, such as the loss of contiguous memory accesses, the decreasing of instruction level parallelism, instruction pre-fetching, etc.;
- the *introduction of overheads* due to additional operations (with respect to the pure MPI parallel approach) for defining the sub-domain shape, for scheduling the tasks to the threads, for balancing the workload, etc.;
- the *ease of code maintenance*;
- the use of a *simple developer interface* for reducing the complexity of the implementation of the new approach; this is important since many developers contribute to NEMO and they will need to follow the selected OpenMP approach when they update or introduce new code.

The final approach has to represent a good tradeoff between the efficiency and the programmability aspects.

2.1 Outer loop parallelisation

The outer loop strategy (OMP1) is based on the parallelisation of the outermost loop of any loop nest. Each computational loop will be parallelised by simply including the OMP DO statement before the outer loop. This means that all of the triply nested loops will be parallelised in the k (vertical/depth) dimension. The doubly nested loops, mainly dealing with the state variables defined on the sea surface and ocean floor, will be parallelised in the j (latitude) dimension. The resulting domain decomposition is depicted in Figure 1. Each thread takes the same horizontal domain assigned to the MPI process (of extent $jpi * jpj$) and a portion of the vertical levels.

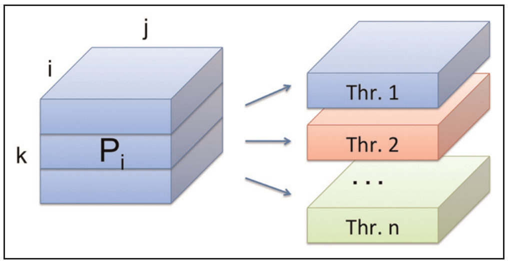

Figure 1. Domain decomposition among OpenMP threads for the outer-loop parallelisation strategy.

From the programmability point of view, this approach is easy to implement, has minimal impact on the original code and is very simple to follow for climate developers (who are not computational scientists). From the efficiency point of view, there is a low level of overhead consisting of some implicit operations for scheduling the loop iterations among the available threads and the thread synchronisation. Finally, each thread still accesses the array elements in memory in the same way as in the original version of the code keeping the same rate of cache hit and level of single instruction multiple data (SIMD) vectorisation. However, from the efficiency point of view, a poor level of parallelisation can be reached. Since the number of iterations along the k-direction (i.e. the number of vertical levels) is typically never greater than 100, this can severely limit the number of threads per process in those architectures with a high number of cores per node (e.g. accelerator-based architectures). The number of iterations in the j-direction can reach 2000, but there are fewer doubly nested loops than triply nested. Moreover, this approach suffers from a poor tuning of the load balance since the grain size consists of horizontal slices containing $jpi * jpj$ grid points.

2.2 3D tiling parallelisation

The second approach (OMP2) is based on the idea of decomposing the MPI domain into a set of 3D tiles. Each tile is characterised by three pairs of indices that define the shape of the tile. The indices also specify the position of the tile within the MPI domain. Thus, each MPI process has to perform the operations for all the tiles belonging to its sub-domain. The OpenMP parallelisation will then be introduced so as to simultaneously perform computation on the tiles assigned to an MPI process. This approach implies that during the initialisation, each process must create its tiles according to a pre-defined domain decomposition strategy. The resulting domain decomposition is illustrated in Figure 2.

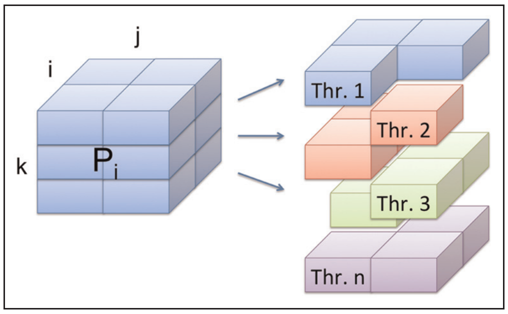

Figure 2. Domain decomposition among OpenMP threads for the 3D tiling parallelisation (OMP2).

In this approach to hybrid parallelisation, the code modifications are quite intuitive and easily adopted by other developers. The 3D tile parallelisation has been implemented using the omp for directive with a static scheduling policy over the tile loops. The static scheduling policy divides the iteration space into contiguous chunks that are approximately equal in size, and one chunk is distributed to each thread. Since the 3D tiles are numbered by row, column and level, a contiguous block of iterations implies also a contiguous block of tiles assigned to a thread. On the other hand, a dynamic scheduling is not required since the decomposition divides the domain in equal parts and the workload is fairly distributed among the threads. From the efficiency point of view, a high level of parallelism can be reached with this approach, i.e. the maximum number of threads can, theoretically, be equal to the number of grid points in the MPI domain (when the tile consists of just one element). Moreover, there is high flexibility in defining the dimensions of the tiles; this can be used to better tune the algorithm and to balance the workload. Even if the workload can be evenly distributed among the threads, it is not always easy to achieve a good load balance: ad-hoc algorithms and strategies must be implemented to define the shape of tiles.

However, there are drawbacks to this approach; the balancing cannot be guaranteed for those loops that do not include all of the elements of the MPI domain (such as those loops that exclude the first or the last vertical level). Moreover, in this approach the data are not contiguously accessed in memory, hence the cache hit-rate decreases and the degree of SIMD vectorisation is worse than in the original code. The ordering of the array indices cannot be changed to alleviate this since the 3D tile cuts the memory in all three directions. Some software overhead is also introduced in defining the tile shapes and in handling the domain boundaries. Changing the tile shapes with the OMP2 implementation we can obtain also the other OpenMP approaches. Actually with the OMP2 approach we slice the MPI domain along three directions (longitudinal, latitudinal and vertical). If the slice is done only along the vertical direction we obtain the OMP1 approach; if the tile is made of only one element we obtain the OMP3 approach described in the next subsection.

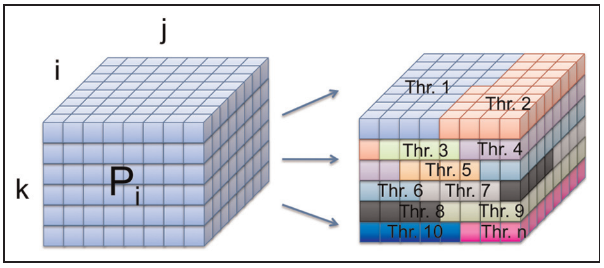

Figure 3. Domain decomposition among OpenMP threads for the loops-merge parallelisation (OMP3).

2.3 Loops merge parallelisation

The last approach (OMP3) is based on the idea of flattening the triply and doubly nested loops such that they become single loops iterating over the total number of elements of the original, nested loop. In other words, it is possible to create a flat loop by merging together the nested loops using the OMP COLLAPSE directive. Figure 3 illustrates the resulting domain decomposition for this approach.

With this approach a high level of parallelism can be reached. The maximum number of threads can, theoretically, be equal to the number of elements in the MPI domain. Moreover, a good load balance can easily be obtained, even for those cases when the whole MPI domain is not swept. Finally, in this approach each thread still accesses contiguous memory locations. However, due to the finer-grained decomposition, this approach could introduce a high level of overhead in some implementations of the OpenMP library. In these cases it could be more efficient to collapse only the outer two levels, especially since this retains the ability to SIMD-vectorise the innermost loop.

3 Array-index ordering

The introduction of further levels of parallelisation with OpenMP and the consequent reshaping of the subdomain assigned to each parallel thread has an impact on both the use of the memory hierarchy and the level of SIMD vectorisation. For these reasons a version of NEMO with the array indices reordered could potentially make more efficient use of hardware capabilities in terms of cache use and instruction-level parallelism. Historically, NEMO and its ancestor, OPA, have been developed for vector architectures as opposed to the scalar architectures of current machines. As a consequence, three-dimensional or rank-three arrays in NEMO are declared with the k (depth) index last, e.g.

$$\mathrm{REAL(wp), DIMENSION(jpi, jpj, jpk) :: b}$$

Therefore, within any sub-domain, the $jpi * jpj$ field values for any level are contiguous in memory. When the target machine comprises a small number of vector processors, $jpi * jpj$ tends to be a large number, which helps effective vectorisation. However, modern machines consist of many thousands of processors and thus $jpi * jpj$ is no longer large. Worse still, it is the loop over i (longitude) that is the innermost of any loop nest and is therefore typically automatically vectorised by the compiler in order to make use of SIMD operations. Thus, as jpi shrinks (as a domain is decomposed over greater numbers of MPI processes) there become fewer SIMD operations available and the efficiency of the code on the CPU drops.

Codes that are functionally similar to NEMO but written with scalar machines in mind (e.g. POLCOMS (Ashworth et al., 2004; Holt et al., 2009)) tend to use an array-index ordering in which it is the levels in a water column that are contiguous in memory. It is likely that several such columns from different arrays can be accommodated in cache at the same time, reducing memory-bandwidth requirements. This layout is also favourable for the halo-exchange operations, because packing and unpacking message buffers now involve copying contiguous blocks of memory. In this ordering it is now the loop over vertical levels that is innermost and the trip count of this remains unaffected by the MPI–OpenMP domain decomposition. This should allow for the code to make more efficient use of the CPU, even when the sub-domains become small.

An experimental version of NEMO has been developed (Pickles and Porter, 2012) in which the choice of array-index ordering is a compile-time option, thanks to the ftrans pre-processor (Booth, 1996). This enables us to investigate the effect of the choice of array-index ordering on the introduction of OpenMP within NEMO.

4 Test cases

Recently, mini-applications (Aloisio et al., 2013), as lighter versions of complex high-performance computing (HPC) applications, have been used as a flexible test bed to facilitate software development. They can be used, as representative of the behaviour of the complete application, to understand the benefits deriving from a proposed software update without changing the entire code. The development process of a mini-application starts from the identification of the bottlenecks of the real application. The computational kernels are extracted and a stand-alone application will be developed for each of them preserving the computational characteristics, the data structures and the communication pattern of the original model.

The NEMO code has been profiled while running the PELAGOS025 configuration (Epicoco et al., 2016; Vichi et al., 2015). PELAGOS025 is a current scientific production configuration in which NEMO is coupled with the BFM biogeochemistry model. It uses a spatial resolution of 1/4 (which generate a spatial grid made of 1442×1021 grid points and 72 vertical levels). The PELAGOS025 configuration has been been profiled on both the Athena parallel cluster based on Intel Xeon processors and on the Vesta BG/Q machine (see next Section for the detailed description of the clusters). The profiling results are reported in Table 1

As with many codes in this domain, NEMO has a broad, flat execution profile with no single routine accounting for more than 15% of run time. There is, therefore, no single bottleneck routine to optimise; for this reason, we have chosen to examine some kernels that typify the common operations that the code performs. We have extracted from the NEMO code two different schemas used for modelling the advection of the tracers (temperature and salinity) and, starting from

Table 1. Code profiling on BlueGene/Q and Intel Sandy Bridge. The data have been taken with the gprof tool and they refer to the timing of the most computationally loaded process on a run with 2048 processes.

BlueGene/Q – VESTA		Sandy Bridge – ATHENA	
Routine	Time (%)	Routine	Time (%)
calchplus	7.70	tra_ldf_iso	12.36
flux_vector	7.25	flux_vector	11.36
tra_ldf_iso	4.36	tra_adv_muscl	10.16
trc_trp_bfm	3.96	trc_trp_bfm	9.69
tra_adv_muscl	3.71	calchplus	7.99
microzoodynamics	2.95	tra_zdf_imp	7.07
mesozoodynamics	2.82	mesozoodynamics	4.1
tra_zdf_imp	2.32	microzoodynamics	3.79
phytodynamics	1.48	pelglobaldynamics	3.42
calcmean_bfm	1.20	phytodynamics	2.95
pelglobaldynamics	1.05	calcmean_bfm	2.64
__lseek_nocancel	16.16		
__read_nocancel	13.16		

Table 2. List of computing platforms.

	Calypso	Vesta	Hector	Athena	PLX	C2A	KNC
Cores/node	32	16	32	16	12	32	61
Hardware Threads	2	4	1	2	2	4	4
Operations per cycle	4	8	8	8	8	8	16
GFlop/node	601.6	204.8	588.8	332.8	230.4	972.8	1010
Clock (GHz)	4.7	1.6	2.3	2.6	2.4	3.8	1.09
Memory/node (GB)	128	16	32	64	48	64	8

these kernels, we have implemented two mini-applications. In common with the NEMO code as a whole, these tracer-related kernels are memory-bandwidth bound due to the large number of array accesses required (primarily for reading). This situation is not helped by NEMO's historical development for vector processors since this has encouraged the use of arrays for storing intermediate results. Writing and reading these arrays use up memory bandwidth that in some cases could be saved by simply re-computing the results as required.

The code of the mini-applications, developed for each kernel, includes a section for the allocation and initialisation of the arrays, a loop which mimics the integration time loop and the kernel itself. The OpenMP Parallel Region has been embedded around the time-loop, therefore the threads are instantiated at the beginning and destroyed at the end. The Parallel Region does not include the initialisation and allocation phase; the arrays are then allocated only by the master thread.

tra_adv_muscl The profiling experiments revealed (Table 1) that the Monotonic UpStream for Conservative Laws advection scheme (MUSCL; the `tra_adv_muscl` routine) accounts for a significant fraction of wall-clock time and it was selected for our tests since its basic code structure and the operations involved are representative of the whole code. It includes several triply nested loops for sweeping the spatial values of the state variables and the halo exchange among neighbours MPI tasks following a cross stencil communication pattern.

tra_ldf_iso In addition to the tracer advection routine, we have also investigated the performance of the `tra_ldf_iso` routine which deals with the lateral diffusion of tracers. This has the advantage of being simpler than the other routine in that it contains no halo exchanges.

5 Performance results

5.1 Architectures

For evaluating the various approaches we have used several computing platforms spread among the main climate and data centres. Namely we have used Calypso (at Euro Mediterranean Center on Climate Change, Foundation, Italy (CMCC)) equipped with IBM Power 6 processors and Athena (also at CMCC) which is an iDataPlex cluster with Intel Xeon E5-2670 Sandy Bridge processors. In the UK we used HECToR, a CRAY XE6 architecture with AMD Opteron 6376 processors, and an Intel Xeon Phi KNC (SE10/7120) co-processor hosted at the Science & Technology Facilities Council, UK (STFC). The latter is a pre-production system and results obtained on the publicly released versions of the Phi may differ significantly. The Argonne Leadership Computing Facility (ALCF) provided us with access to an IBM BG/Q machine named Vesta. At CINECA (which is the National Consortium for Advanced Computing in Italy) we used an iDataPlex cluster, named PLX, with the Intel Xeon E5645 Westmere and at ECMWF we used the C2A cluster which is an IBM Power7 based architecture. The main computing characteristics of each machine are reported in Table 2.

5.2 Results

Three kinds of test have been performed. The first set of tests aims to evaluate the suitability of the hybrid parallelisation on different architectures and to identify the optimal ratio of MPI processes to OpenMP threads on each. For these tests, the `tra_adv_muscl` kernel has been considered using two different domains: the first includes about 16 million grid-points and is similar to the Mediterranean Forecast System (MFS) configuration implemented at CMCC. The second domain is defined in order to saturate the available memory on the compute node of each architecture under test.

The second set of tests aims to compare the different architectures with respect to the use of OpenMP parallelisation. For these tests the `tra_ldf_iso` kernel has been used.

The third set of tests aims to evaluate the impact of re-ordering the array indices on both the level of vectorisation and, more generally, the performance. For these tests the `tra_adv_muscl` kernel has been considered. The index reordering tests have been carried out for both the outer-loop and tiling approaches to introducing OpenMP. For these tests, two different grid sizes corresponding to global ORCA grids of 1 and 2 resolution have been used. The latter, denoted ORCA2, has dimensions $182 \times 149 \times 31$ while the former, ORCA1, has dimensions $362 \times 292 \times 46$. These correspond to 800,000 and 4.8 million grid points, respectively.

The following paragraphs report the results obtained on each architecture. The execution time and speed-up charts aim at comparing the pure MPI version with the proposed OpenMP approaches. The OpenMP scalability tests have been executed using only one MPI process per node and a number of threads per process equal to the number of cores on the node. To evaluate the best ratio between MPI processes and OpenMP threads the charts with performance as a function of the processe/thread mix should be analysed. In these the total number of parallel elements is kept constant while the ratio of threads to MPI processes is varied. The solid lines in each chart refer to the first domain, while the dashed lines refer to the biggest one.

As a general consideration, the main issue to be addressed in the first set of tests is to verify if the usage of threads instead of processes can bring some performance improvement. Given a fixed number of cores, how many processes and threads should be instantiated? When the number of threads increases, the number of processes decreases such that the total number of cores in use is kept constant. Increasing the number of threads implies, therefore, that the MPI sub-domains become bigger. There is therefore a reduction in the number of MPI messages while the messages themselves increase in size. However, the use of a large number of OpenMP threads will increase the overheads associated with thread synchronisation.

When the number of threads is reduced, the MPI sub-domains become smaller (since we will have more processes). Therefore, the quantity of MPI messages increases but smaller messages are exchanged (since the MPI sub-domains are smaller), and the overhead due to thread synchronisation decreases. The performance improvement is therefore given as tradeoff between three main factors: MPI communications; thread synchronisation; and data locality (this last aspect can be improved by careful control of thread affinity although in some cases it cannot be imposed).

As an example we consider the OMP1 approach where the thread workload is obtained by distributing the vertical levels. If we fix the total number of processing elements then we fix the product of the number of MPI ranks and the number of threads per process. We can consider two extreme cases; the first when the global domain has a large number of vertical levels and the second when there are few vertical levels.

In the first case it is more efficient to use a high number of threads per MPI rank (hence, few MPI ranks; we recall that the total number of processing elements is fixed) since the workload assigned to each thread would be sufficient to make the synchronisation overhead small relative to the computational time. Moreover, having few MPI ranks also reduces the number of MPI messages, albeit with larger messages (we recall here that each MPI rank exchanges with its neighbours all of the values along the border for all vertical levels). When the number of vertical levels is small, it is more efficient to use just a few threads per MPI rank since the workload assigned to each thread is not large and thus the thread synchronisation overhead is not negligible. Moreover, having more MPI ranks, the horizontal domain is divided into many smaller sub-domains and consequently the MPI communications involve many, small messages. The experimental results given in the following sections fall between these two extremes since they employ the common spatial resolutions used in real climate studies.

A further consideration related to the thread-memory affinity regards the way in which memory is allocated and initialised. The allocation of physical memory under Linux happens at the point where it is first written to (and not at the point at which it is allocated). Usually, before spawning the thread pool, the MPI process (hence, the master thread) allocates and initialises the data structures. At the first touch the data are physically allocated on the memory 'near' to the master thread; the other threads then have to access this data which is allocated 'far' from them with a loss in performance. A better strategy is to spawn the threads also during the initialisation phase such that each thread can initialise its portion of data and hence physically allocate the data in the 'affine' memory. The thread-memory affinity is strictly linked also to the binding between threads and cores. On some architectures the binding of the threads to a core is imposed by default (e.g. on Cray and on BG/Q), on other architectures the binding must be activated explicitly during program execution.

We used the thread binding and thread memory affinity on all of the architectures which efficiently support it. On the Athena cluster for example, the best performance was been obtained when the process' threads were bound to the multi-chip module (MCM), and not to a core; on the PLX cluster it was not possible to control the thread pinning. The first-touch strategy has been implemented in both the `tra_adv_muscl` and `tra_ldf_iso` kernels and has been used in all of the experiments. For all tests the standard MPI v1.2 and OpenMP v3.0 have been used.

IBM Power6 (Calypso at CMCC) On the IBM Power6 at CMCC a large domain of 600 million grid points has been considered. The execution time and the parallel speedup are reported in Figure 4. Here the best performance is obtained with the pure MPI implementation; none of the hybrid parallel approaches brings any improvement. Figure 5 reports the analysis of the

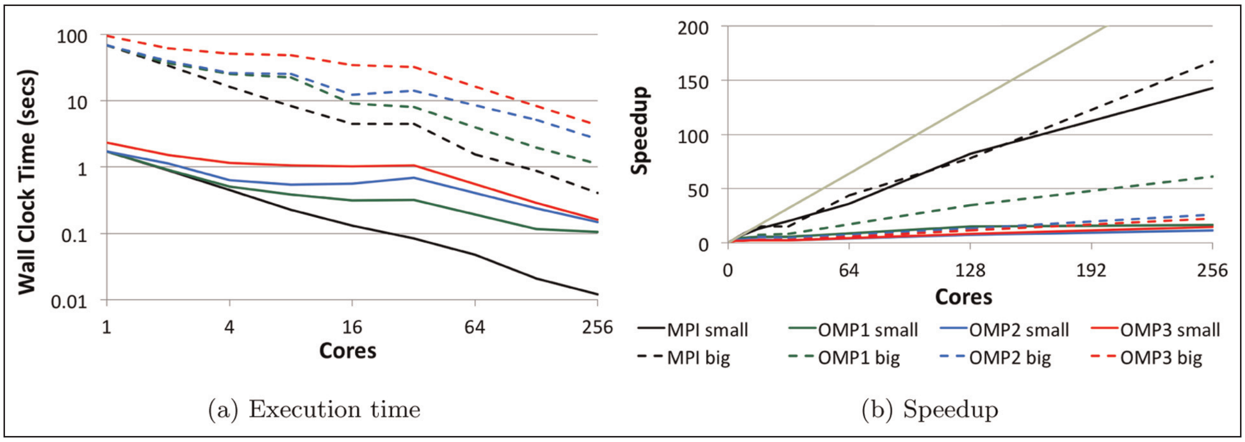

Figure 4. Comparison between pure MPI approach with the proposed OpenMP ones on Calypso cluster.

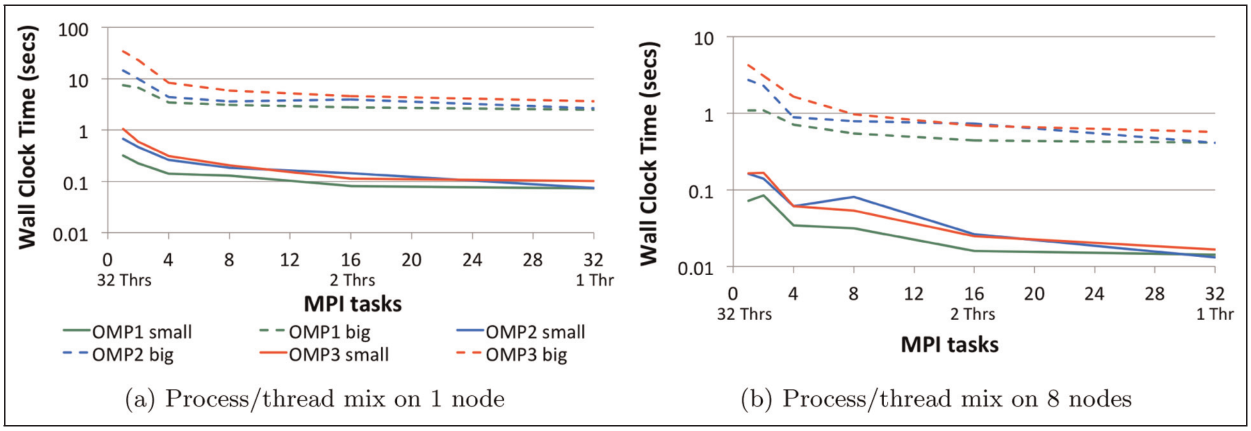

Figure 5. Evaluation of ratio between processes and threads on Calypso cluster.

ratio of MPI processes to OpenMP threads on both a single node and on eight nodes. In both cases the x-axis reports the number of MPI process per node. These charts also demonstrate that the best performance is achieved when using one thread per process. On this machine the MPI implementation fully exploits the node capabilities using shared memory for intra-node communication. Moreover the pure OpenMP run (32 threads per process) introduces an overhead due to the synchronisation point at the end of the parallel do loops so the advantage of a shared memory approach is not appreciated on this architecture.

BG/Q (Vesta at Argonne) Vesta has 64 GB of main memory per node and for this reason a big domain made of 60 million grid points has been used to saturate the memory. The scalability analysis (Figure 6b) shows that the pure MPI implementation scales better than any of the mixed-mode versions. The scalability reaches the limit at 4096 cores for MPI and at 2048 for mixed mode. Figure 7 reports the analysis of the processes/threads mix. For the runs within a single node (Figure 7a), the lines are horizontal, meaning that performance is independent of whether processes or threads are employed. The tile-based parallelisation is an exception since the best results are achieved when using 64 threads per process (i.e. a pure OpenMP implementation). When the computation involves more than one node (Figure 7b) and hence the MPI communications must use the network, the benefit of the OpenMP implementation becomes much more evident.

Compiling a parallel code with OpenMP enabled often hampers the optimisations that a compiler performs. The pure MPI scalability (black lines in Figure 6b) has been obtained using an executable compiled without the OpenMP flag. However, the thread/process mix analysis of Figures 7a and 7b has been done using an executable compiled with the OpenMP flag active and the wall-clock time for 64 MPI tasks (i.e. pure MPI) is obtained by executing the mini-application with OMP_NUM_THREADS = 1, this explains why the pure MPI implementation scales better even if Figure 7b

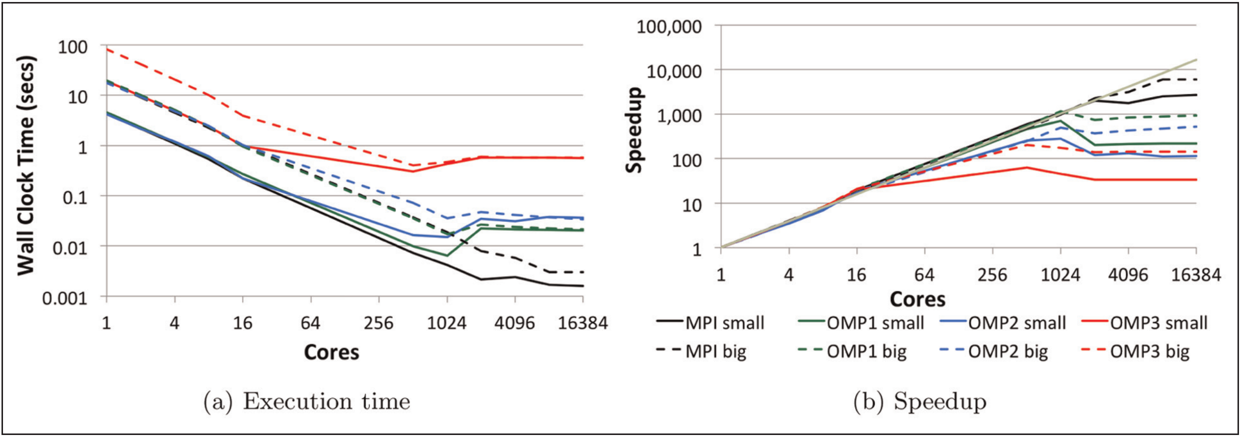

Figure 6. Comparison of the pure MPI approach with the proposed OpenMP schemes on the Vesta cluster.

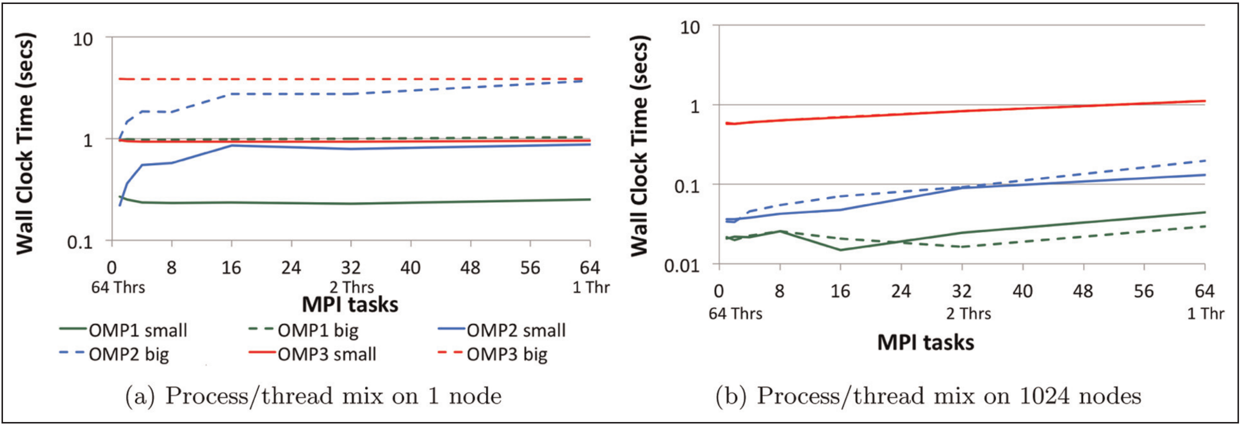

Figure 7. Evaluation of ratio between processes and threads on the Vesta cluster.

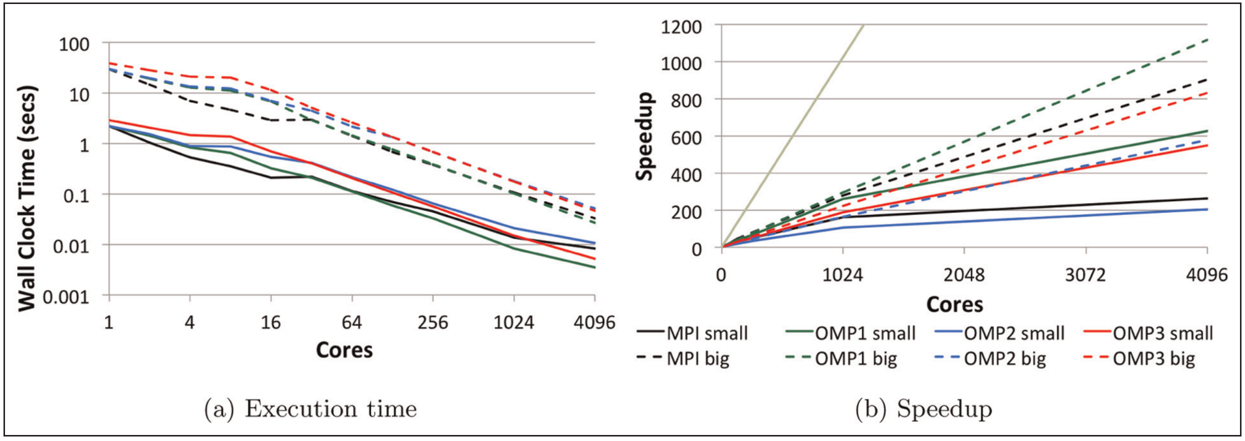

Figure 8. Comparison between the pure MPI approach and the proposed OpenMP schemes on Hector.

suggests that four threads per MPI task is the best mix. Finally, of the OpenMP implementations, the outer loop approach (OMP1) performs the best. On BG/Q we have also ported the whole NEMO code and optimised part of it in a previous work available in Epicoco et al. (2013).

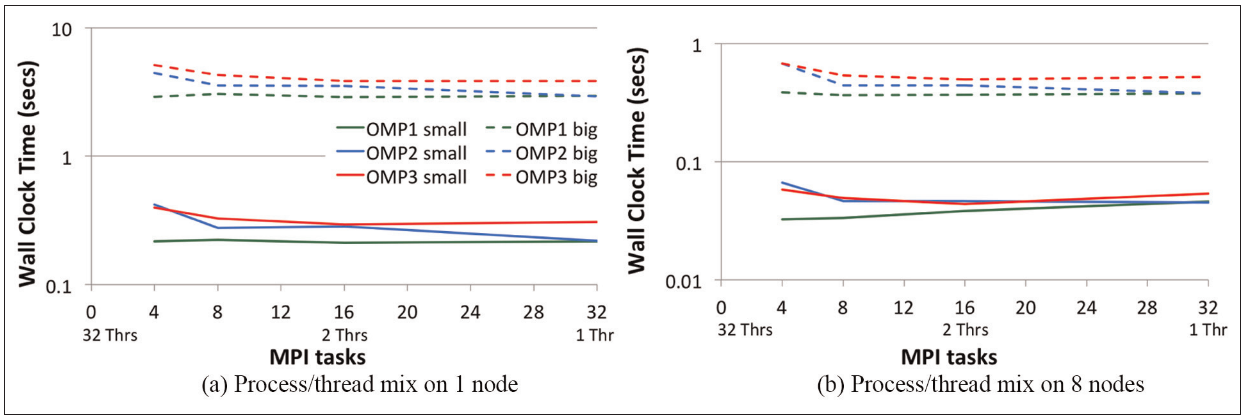

Figure 9. Evaluation of ratio between processes and threads on Hector.

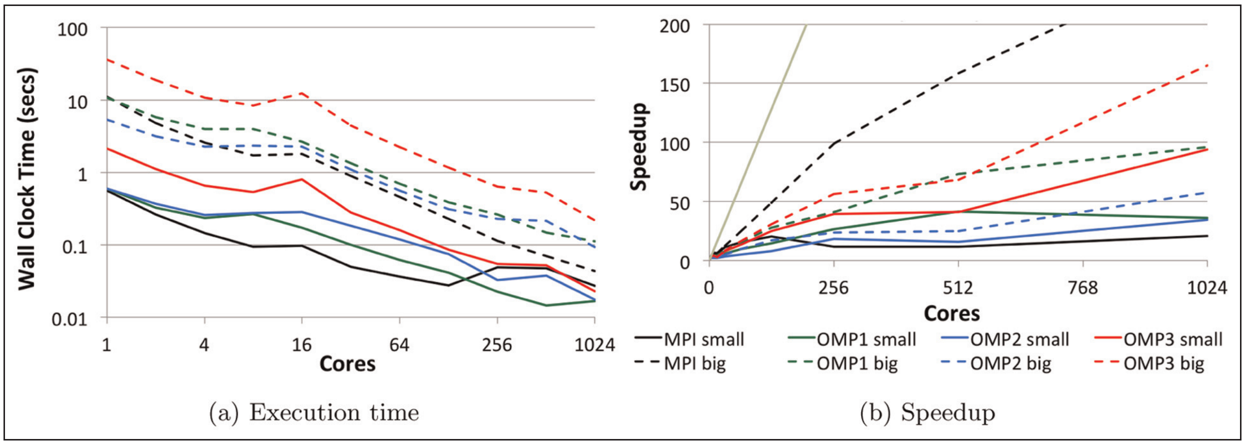

Figure 10. Comparison of the pure MPI approach with the proposed OpenMP schemes on Athena.

Cray XE6 (HECToR, UK) On the Hector machine the analysis has been performed using a large domain with 215 million grid points. As shown in Figure 8a, the OpenMP implementation performs better than the pure MPI for a number of parallel tasks greater than 32. The improvement is most evident with the small domain runs but is also present for the bigger domain. To evaluate the best tradeoff between processes and threads, the analysis has been carried out by using a single node (Figure 9a) and eight nodes (Figure 9b); in both cases, for each node, the product of the number of MPI processes with the number of OpenMP threads per process has been fixed at 32. Figure 9 shows that the outer loop parallelisation (OMP1) performs better than the others and fully exploits the capabilities of the architecture when running with eight threads per process; the tile-based (OMP2) and the loop-merge (OMP3) approaches perform best with four threads per process.

IBM iDataPlex Sandy Bridge (Athena at CMCC) On the Athena machine, we find appreciable benefits to using OpenMP parallelisation after only 120 cores, as shown in Figure 10a. The large configuration chosen for this machine consists of 250 million grid points. Again, the outer loop parallelisation (OMP1) performs better than the others. Finally, the best configuration is given by using four MPI processes per node with four threads for each process and this holds for both runs within a single node (Figure 11a) and for runs with 16 nodes (Figure 11b).

This result can be explained in terms of the internal memory hierarchy of the dx360 node which consists of two sockets with eight cores each. The L1 and L2 caches are private to each core, while the L3 cache is shared among the cores in the socket. Nothing is shared between the two sockets (including the system RAM). With this architecture we lose performance every time a thread is migrated to a different core, even within the same socket, since this invalidates all of the L1 and L2 cache entries. Even worse is the case when the thread migration occurs between cores belonging to different sockets because, in this case, not only are the L3 cache entries lost but also the thread must then access the remote memory (i.e. the DRAM connected to the other socket) using the QPI (QuickPath Interconnect)

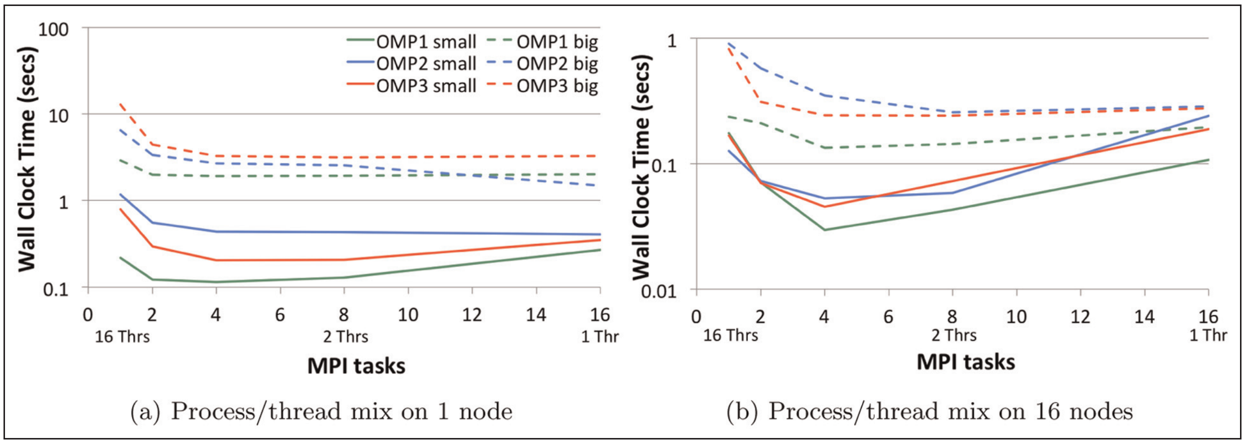

Figure 11. Evaluation of ratio between processes and threads on Athena.

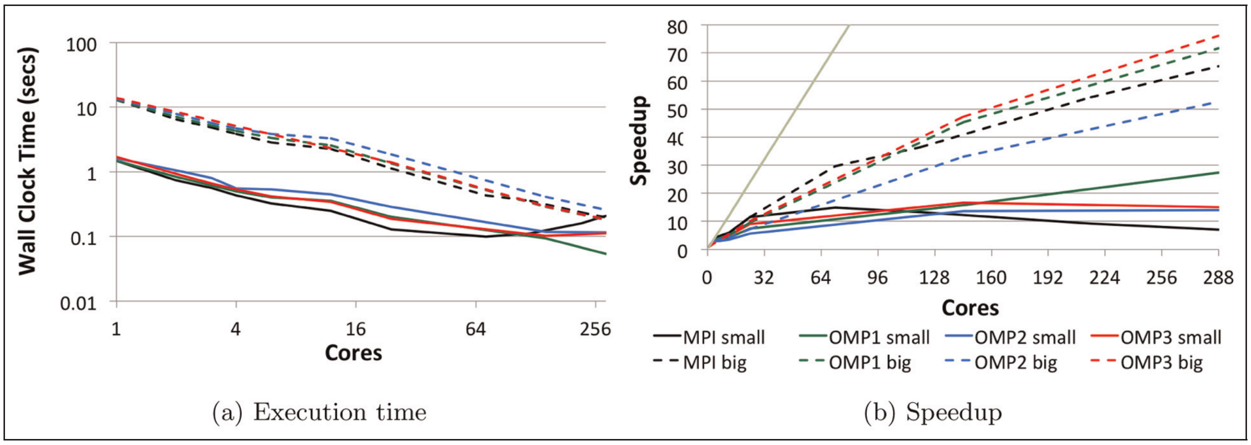

Figure 12. Comparison of the pure MPI approach with the proposed OpenMP schemes on the PLX cluster.

internal network on the motherboard. In the pure OpenMP run (16 threads per process) the data structure is allocated by the master thread in the main memory local to the first socket, the threads mapped onto the second socket access the data through the QPI network from the first socket resulting in a significant loss of performance. In this experiment, the task affinity has been set to *core* for the pure MPI run and to *MCM* (Multi Chip Module) for all of the other cases; no thread affinity has been imposed, hence the threads are allowed to migrate among the cores assigned to the process. For runs within a node the overhead due to thread synchronisation is greater than the message passing overhead. However, once several nodes are employed and MPI must communicate between nodes then this is no longer the case.

IBM iDataPlex Westmere (PLX at Cineca) On the PLX machine, the OpenMP implementations perform better than the pure MPI version on more than 120 cores. For saturating the main memory a large domain of 140 million grid points has been used. Again in this case the outer loop approach (OMP1) performs better than the other options as shown in Figure 12a. The analysis of the best tradeoff between processes/threads shows that the optimal configuration is given by using two MPI processes per node with six threads per process (see Figure 13). For the runs within a single node the lines are mostly horizontal (see Figure 13a) although there is a shallow minimum at six threads per process. The runs with 24 nodes have some jitter due to the interprocess communications but again the best tradeoff is obtained with six threads per process.

The results can be understood if we consider that a node consists of two sockets with six cores each. When we allocate 12 threads to a single MPI process, the execution time increases. This is because the threads executed on the second socket will access memory through the first socket, dramatically increasing the memory latency. Use of thread pinning would almost certainly

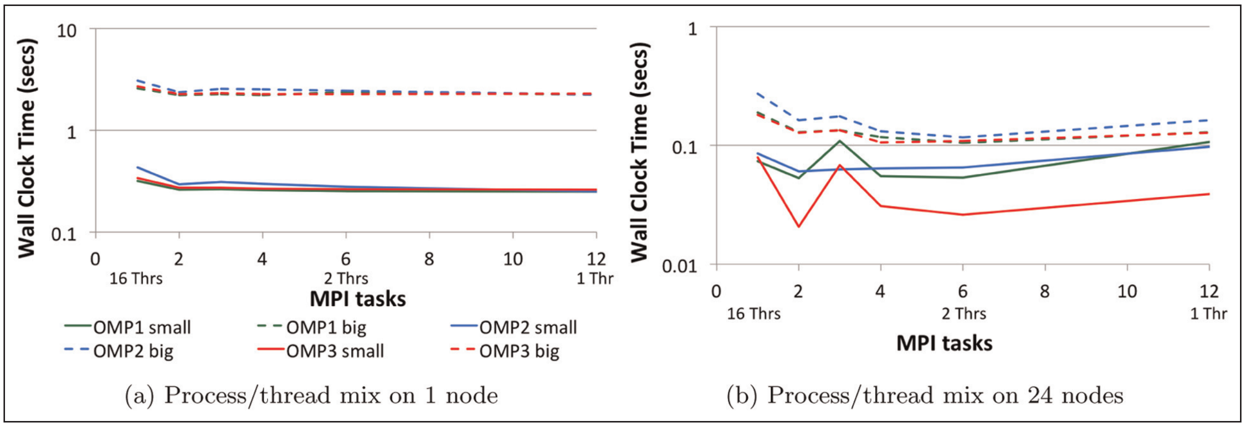

Figure 13. Evaluation of the best ratio of processes to threads on the PLX cluster.

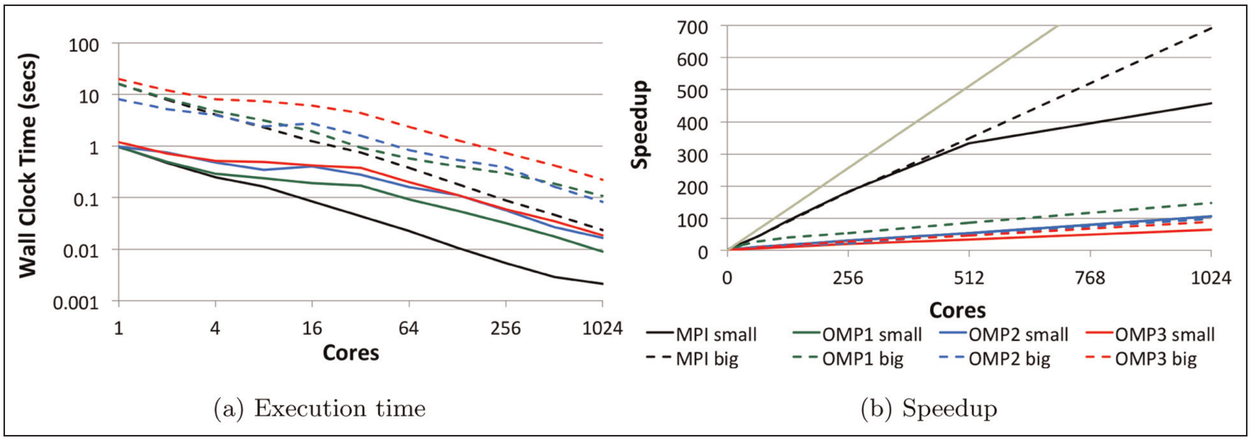

Figure 14. Comparison of the performance of the pure MPI approach with the proposed OpenMP schemes on the C2A cluster.

improve performance but we were unable to enable it on this machine.

IBM Power7 (C2A at ECMWF) The IBM Power7 is the successor to the Power6 and is based on the same computing architecture with several performance improvements. On this machine the large domain tests are based on a configuration with 250 million grid points. As with the Power6 system considered earlier, none of the hybrid parallel approaches improves on the performance of the pure MPI approach as demonstrated by the plot in Figure 14a.

To summarise, pure MPI was found to perform best on both of the IBM Power-based machines. On the remaining machines best performance was obtained from mixed-mode versions of the application. In every such case, the most performant version was found to be that where only the outer loop (over vertical levels) was decomposed over OpenMP threads. The precise details of the best-performing configuration (in terms of number of threads per MPI process) are highly dependent on the architecture of the nodes making up a machine. In general, it is best to ensure that an OpenMP thread team are executing within a single NUMA region. For example, the nodes of the PLX machine at Cineca consist of two Westmere sockets, each with six cores. It is therefore best to have a team of six threads on each socket and to use MPI between sockets.

5.3 Architecture comparison

In this set of tests our aim is to provide some insight on the main multicore architectures available today in HPC clusters when running the pure OpenMP version of the most simple of the kernels, `tra_ldf_iso`. For this kernel we implemented only the outer-loop OpenMP approach since it has been demonstrated to be more performant. The comparison has been made between the following architectures: the AMD Magny Cours; IBM Power7; Intel Westmere, Intel Sandy Bridge; and Intel Xeon Phi (Knights Corner). In the

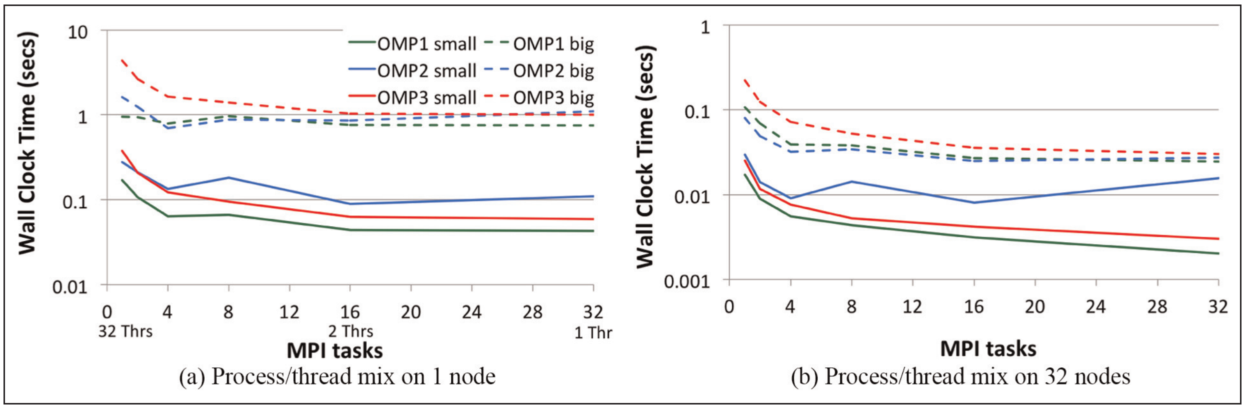

Figure 15. Evaluation of the best ratio of processes to threads on the C2A cluster.

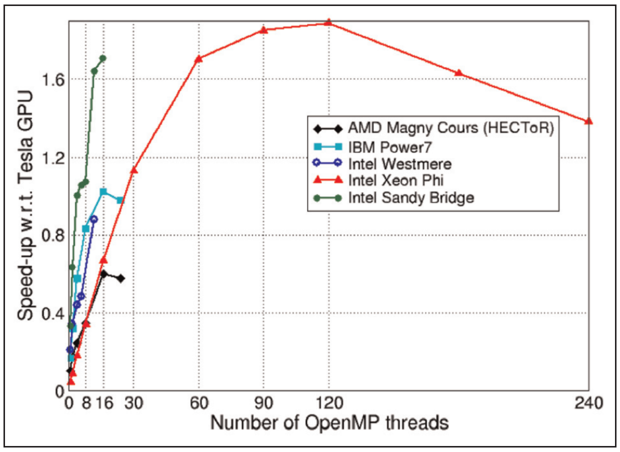

Figure 16. Performance of the `tra_ldf_iso` kernel with outer loop OpenMP parallelisation for the ORCA2 grid. Performance is plotted relative to that obtained previously on an NVIDIA Tesla M1060 GPU (Porter et al., 2012).

experiments we executed the mini-application with only one MPI task while changing the number of threads. The results reported in Figure 16 demonstrate that the best absolute performance was obtained with 120 threads (i.e. two threads per core) on the Intel Xeon Phi. On more than 120 threads performance drops off rapidly in this case. A close second in terms of performance is the dual-socket Intel Sandy Bridge system. In this case the performance of the code continues to improve out to 16 OpenMP threads, occupying the full 16 hardware cores of the system. However, a plateau in the curve can be seen in the region of eight threads as the memory bandwidth to a single socket becomes exhausted. The most dramatic feature of Figure 16 however is the increase in performance obtained in going forward one CPU generation from the Intel Westmere to the Intel Sandy Bridge. On a single socket, the `tra_ldf_iso` kernel runs slightly more than twice as fast on the Sandy Bridge as it does on the Westmere.

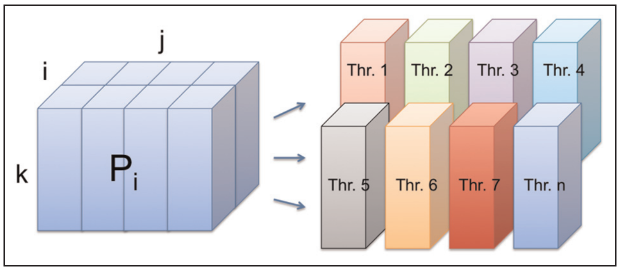

Figure 17. Domain decomposition among OpenMP threads for the 2D tiling parallelisation.

On two sockets this factor has decreased slightly but the Sandy Bridge is still almost twice as fast. Note that we obtained optimum performance on the Xeon Phi when using the *balanced* binding of threads to hardware cores (KMP_AFFINITY = balanced) and *dynamic* loop scheduling (OMP_SCHEDULE = dynamic). As expected, the performance of the Xeon Phi is very sensitive to load balance when using large numbers of threads.

5.4 Evaluation of array index reordering

The last set of tests aims to evaluate the impact on performance of the array index reordering. For these tests the `tra_adv_muscl` kernel has been used, comparing the outer-loop and 2D tile-based parallel approaches. In contrast to the OMP1 and OMP3 approaches described earlier, here we collapse (flatten) only the outer two loops of a triply nested loop. For the 2D tiling, each MPI sub-domain is divided into tiles but only in the *i*,*j* (latitude–longitude) plane and these tiles are shared amongst OpenMP threads. Each tile retains the full domain extent in the third dimension. This decomposition can also be obtained using the OMP2 approach and slicing only along the longitudinal and latitudinal directions. This idea is illustrated in Figure

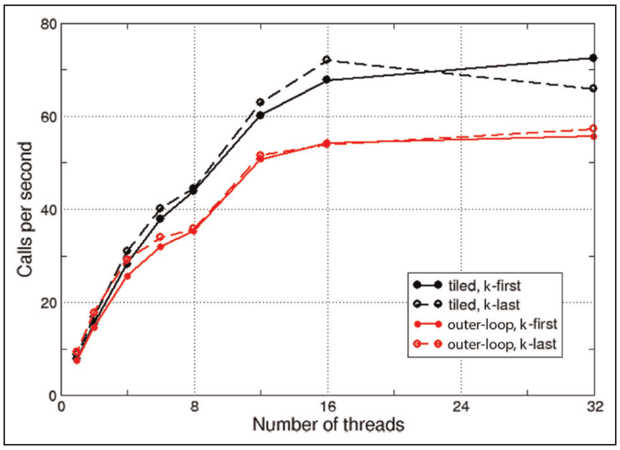

Figure 18. Performance of the tra_adv_muscl kernel on Intel Sandy Bridge for the ORCA2 grid. The number of times per second it was possible to execute the whole kernel is reported along the *y*-axis. Hyperthreading is only used for the 32-thread results. Here *k*-first indicates that array indices have been re-ordered such that the level/vertical index is fastest varying.

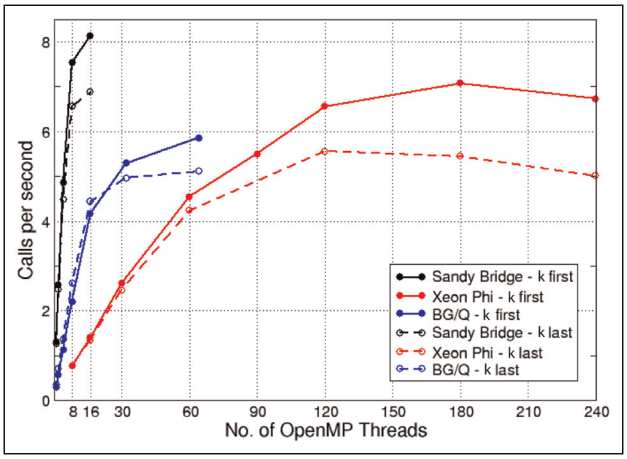

Figure 19. Performance comparison of the tiled version of tra_adv_muscl on Intel Sandy Bridge, Intel Xeon Phi and IBM BG/Q (single socket of each). The number of times per second it was possible to execute the whole kernel is reported along the *y*-axis. The ORCA1 grid has been used.

17 where the MPI sub-domain is shown decomposed into eight tiles.

The advantage of these approaches is that they retain the innermost loop of any triply nested loop for a thread to work on. We do this to utilise the instruction-level (SIMD) parallelism common on current CPU architectures. As an example, Intel's Sandy Bridge core supports 256-bit SIMD operations (so that it is capable of working on four, double-precision floating point variables simultaneously) while the core on the Intel Xeon Phi supports 512-bit SIMD. We rely on the compiler's auto-vectorisation capability to parallelise the innermost loop using these SIMD instructions. By not restricting the number of tiles to be the same as the number of threads it becomes possible to tune the tile size so as to make better use of on-chip cache. Having more tiles than threads can also improve the load balance between threads as the scheduling of work can be handled by the standard OpenMP *dynamic* or *guided* thread scheduling.

The results for the ORCA2 configuration on the Sandy Bridge system are shown in Figure 18.

For the tile-based version, the plotted results are for the best tiling grid found for each thread count. At low thread counts there is little difference between the two versions although the outer loop version performs slightly better. However, on four or more threads the tile-based implementation is clearly best; the advantage growing with the number of threads. As already observed, there is a clear plateau in the performance plot for the outer loop implementation as the first socket becomes full at eight threads and memory bandwidth is saturated. This dip in performance is much reduced with the tile-based version which is some 26% faster than the outer-loop version on eight threads. We attribute this increase in performance to an improved use of cache and thus of available memory bandwidth.

Figure 18 also shows the performance of the `tra_adv_muscl` kernel where the array indices are re-ordered such that the *k*-index is first (indicated by the solid lines and symbols). On eight threads (a full socket), this switch in ordering has no effect on performance, consistent with the idea that memory bandwidth is exhausted in this case. However, on the majority of thread/core counts, the *k*-last implementation out-performs the *k*-first. This is particularly evident in the tiling form of the kernel. In this case we found that for the *k*-last ordering it was always optimal to use a tiling grid which only subdivided the second (*j*) dimension. This left the full extent of the *i* dimension available for SIMD vectorisation. With the *k*-first ordering, the trip count of the loop to be SIMD vectorised is 31 (i.e. the number of vertical levels), much less than the *i* dimension of the ORCA2 grid (182).

In Figure 19 we compare the performance of the `tra_adv_muscl` kernel on a single socket of Sandy Bridge, Xeon Phi and PowerPC A2 (from the IBM BG/Q) for the larger, ORCA1 grid.

On all of these chips the use of simultaneous multithreading (IBM)/hyperthreading (Intel) can be seen to give a considerable performance improvement. The Intel Sandy Bridge gives the best performance although the Xeon Phi is not far behind. Unlike the ORCA2 case, the version of the kernel with *k*-first ordering now consistently gives best performance on higher thread counts and enables the Xeon Phi to scale out to 180 rather than 120 threads. The increase in the number of vertical levels from 31 to 46 is likely to be significant in this change in behaviour.

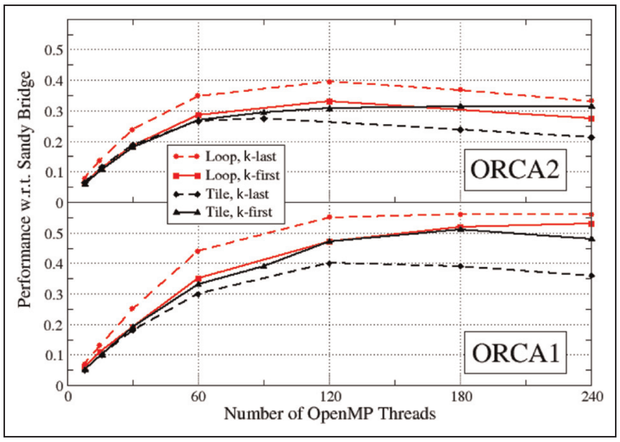

Figure 20. Performance comparison of the outer-loop and tiled versions of tra_adv_muscl on the Intel Xeon Phi. The top plot is for the ORCA2 grid and the bottom is for the larger, ORCA1 grid.

Table 3. Summary showing the most performant version of the `tra_adv_muscl` kernel for the ORCA1 grid on each architecture.

	OpenMP parallelism	Array-index ordering
Sandy Bridge	Tiled	*k*-first
Knights Corner Phi	Loop collapsed	*k*-last
A2	Tiled	*k*-first

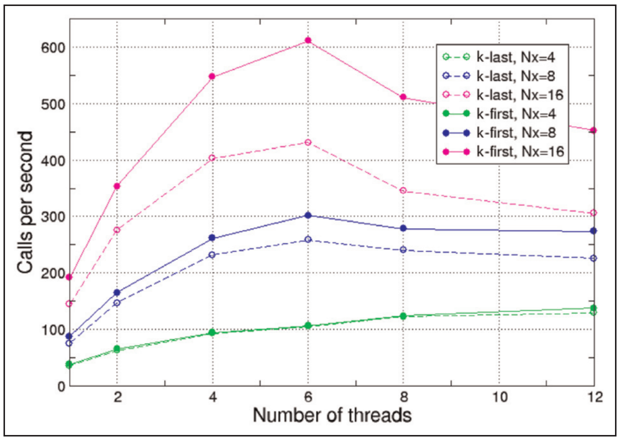

Figure 21. Effect of simulated MPI decomposition on computational performance of the tra_adv_muscl kernel in the outer loop implementation with the ORCA2 grid. Here N_x is the extent of an MPI processor grid in the *i* dimension. Kernel runs on a single node of HECToR. The number of times per second it was possible to execute the whole kernel is reported along the *y*-axis.

Although we have plotted performance for the tile-based version of the kernels in Figure 19, this is actually unfair on the Xeon Phi since it turns out that the outer-loop implementation performs better on that hardware. This is shown in more detail in Figure 20 where we compare the performance of the Xeon Phi with the full, dual-socket Sandy Bridge system.

It is clearly the outer-loop parallelisation, *k*-last form of the kernel, that performs best for both the ORCA1 and ORCA2 grids. However, up to 60 threads the remaining three alternatives perform very similarly. We attribute this to a combination of the advantage of a longer inner loop (for SIMD vectorisation) and the fine-grained parallelism permitted by allowing OpenMP to distribute work at the outer loop (and consequently achieve a better load balance). The importance of the trip count of the (SIMD-vectorised) inner loop is seen in the performance gap between the *k*-first and *k*-last loop-level results on low thread counts. In the *k*-last case it is the full *i* dimension that is SIMD-vectorised and this is considerably greater than the number of vertical levels. On larger thread counts, the additional trip count of the outer loops (over *i* and *j*) in the *k*-first case allows for better load balance and performance scales out to 240 threads for the ORCA1 grid, almost matching the *k*-last version. The increased amount of parallelism in the ORCA1 grid clearly benefits the Xeon Phi as it achieves approximately 55% of the performance of the (two-socket) Sandy Bridge system, as opposed to 40% with the ORCA2 grid.

Finally, in order to emphasise the potential benefits of the *k*-first index ordering, Figure 21 shows the effect on the computational performance of the outer loop version of the `tra_adv_muscl` kernel when decomposing the ORCA2 grid over a regular grid of MPI processes.

This processor grid is assumed to consist of $N_x \times N_y$ processors so that the *i* dimension of the ORCA2 grid is divided into N_x sub-domains. This then leads to a reduction in the *i* dimension of the arrays that each process has to work upon. For a small processor grid (mimicked with $N_x = 4$), the trip count of the innermost loop will be $182/4 = 45$ in the *k*-last ordering. This is greater than the trip count of 31 (vertical levels) in the *k*-first ordering and consequently there is little difference in the performance of the two kernels. However, for $N_x = 8$, the trip count falls to 22, which is 30% less than the number of vertical levels and the *k*-first version performs better (by about 20% in this case). At $N_x = 16$ the trip count in the *k*-last version is just 11 and the *k*-first version performs significantly better (about 50%).

We summarise these findings in Table 3. For the traditional multi-core CPUs that we experimented with, best performance was obtained by switching to a *k*-first array index ordering and decomposing the *i*,*j* iteration space into tiles. However, for the many-core Xeon Phi, best performance was obtained by leaving the kernel code unchanged and using an OpenMP directive to collapse the outer two loops. This is primarily due to the Phi's longer SIMD registers and the resulting sensitivity to the length of the SIMD-vectorised inner loop. With the current trend for increasing SIMD vector widths, even on traditional CPUs, this sensitivity to the amount of SIMD vector work is only set to increase.

6 Conclusions

We have investigated a variety of approaches for introducing hybrid parallelism into the NEMO ocean model. Our experiments with individual, tracer-related kernels have shown that best absolute performance is obtained by simply parallelising the outer loop with OpenMP. Good practice when considering thread-based parallelism, beyond the issues of load balancing and thread scheduling, concerns the optimal management of memory accesses and cache reuse. The best performance can be achieved when (i) the thread operates on a contiguous data set (an optimal domain decomposition can guarantee this property); (ii) the thread is bound with the core using memory affinity; (iii) false sharing is avoided (the mechanism whereby a cache line of a thread is invalidated by another thread which accesses different data which belong to the same cache line).

We have also shown that the change of array index ordering from k-last to k-first can provide a significant increase in performance in the strong-scaling limit when MPI sub-domains become relatively small. However, this gain comes at the cost of a massively intrusive and pervasive change to the NEMO code base. Since that code base is under continuous development, the benefits of such a change must be carefully considered (Pickles and Porter, 2012).

The Intel Xeon Phi struggled to match the performance of the traditional Xeon CPU as soon as the kernel involved the thread synchronisation and load imbalance introduced by halo-exchange (MPI) operations. The results also demonstrate that, for the traditional architectures, the management of multiple threads within a node is less efficient than the management of Linux processes and the pure MPI implementation is good when the sub-domain is relatively big. The hybrid implementation outperforms the pure MPI version once the sub-domain size becomes small on both the Intel and IBM BG/Q systems. However, on the IBM Power-based systems the pure MPI implementation was the most performant in all of our tests. Finally, considering all of the proposed approaches, we conclude that the introduction of OpenMP through the parallelisation of the outer loop (over vertical levels) can be considered the best tradeoff between programmability, robustness and performance.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was partially founded by EU Seventh Framework Programme within the IS-ENES project [Grant number 228203] and IS-ENES2 project [Grant number 312979] and by the PRACE Second Implementation Phase Project (PRACE-2IP) under call FP7-INFRASTRUCTURES-2011-2 [Project number 283493].

References

Aloisio G, Andrè JC, Epicoco I and Mocavero S (2013) The role of mini-apps in weather and climate models performance optimisation. In: *White Paper in 2nd Big Data Extreme Computing BDEC International Workshop*.

Ashworth M, Holt J and Proctor R (2004) Optimization of the POLCOMS hydrodynamic code for terascale high-performance computers. In: *Proceedings of the 18th International Parallel and Distributed Processing Symposium, 2004*. DOI:10.1109/IPDPS.2004.1302910.

Booth S (1996) FTRANS user guide. Available at: http://www.nzdl.org/gsdlmod?e = d-00000-00—off-0cstr–00-0—-0-10-0—0—0direct-10—4——-0-1l–11-en-50—20-about—00-0-1-00-0–4—-0-0-11-10-0utfZz-8-00&cl = CL1.168&d = HASH014b5e04a0f85ac7e6-c55e00.1&x = 1 (accessed 7 December 2016).

ECMWF (2015) IFS documentation CY41R1 operational implementation - part VI: Technical and computational procedures. Availabe at: http://www.ecmwf.int/sites/default/files/elibrary/2015/9213-part-vi-technical-and-computational-procedures.pdf (accessed July 2016).

Epicoco I, Mocavero S and Aloisio G (2013) Porting and performance analysis of the NEMO ocean model on Blue Gene/Q. In: *Extended abstract in ScicomP/SPXXL Workshop*.

Epicoco I, Mocavero S, Macchia F, et al. (2016) Performance and results of the high-resolution biogeochemical model PELAGOS025 V1.0 within NEMO V3.4. *Geoscientific MODEL Development* 9: 2115–2128.

Holt J, Harle J, Proctor R, et al. (2009) Modelling the global coastal ocean. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 367(1890): 939–951.

Leopardi P (2006) A partition of the unit sphere into regions of equal area and small diameter. *Electronic Transactions on Numerical Analysis* 25: 309–327.

Madec G and the NEMO Team (2008) NEMO ocean engine. *Note du Pole de modellisation*, Institut Pierre Simon Laplace.

Michalakes J, Dudhia J, Gill D, et al. (2005) The weather research and forecast model: Software architecture and performance. In: *Proceedings of the Eleventh ECMWF Workshop on the Use of High Performance Computing in Meteorology*, pp. 156–168.

Pickles SM and Porter AR (2012) Developing nemo for large multi-core scalar systems: final report of the DCSE NEMO project. DL Technical Reports DL-TR-2012-002.

Porter A and Ashworth M (2010) Configuring and optimizing the weather research and forecast model on the Cray XT. In: *Cray User Group Proceedings*.

Porter AR, Pickles SM and Ashworth M (2012) Final report for the GNEMO project: porting the oceanographic model NEMO to run on many-core devices. DL Technical Reports DL-TR-2012-001.

Smith L and Bull M (2001) Development of mixed mode mpi/openmp applications. *Scientific Programming* 9(2–3): 83–98.

Smith R, Jones P, Briegleb B, et al. (2010) The Parallel Ocean Program (POP) reference manual. *Report LAUR-10-01853*. Available at: http://www.cesm.ucar.edu/models/cesm1.0/pop2/doc/sci/POPRefManual.pdf (accessed July 2016).

Vichi M, Gutierrez Mlot GCE, Lazzari P, et al. (2015) The Biogeochemical Flux Model (BFM): Equation description and user manual, BFM Version 5.0 (BFM-V5), release 1.0. *BFM Report Series 1*, Bologna, Italy. Available at: http://bfm-community.eu/ (accessed 7 December 2012).

Author biographies

Italo Epicoco is an Assistant Professor at University of Salento, Lecce, Italy. He received a PhD in Computational Engineering at the University of Lecce, Italy. He is teaching a class in the Engineering Faculty in “Foundations of Computer Science”. He is an affiliate researcher of the Euro-Mediterranean Center on Climate Change (CMCC). His research interests include high-performance computing (HPC), distributed, grid and cloud computing. During his research activities he has addressed issues related to the optimisation of numerical methods for solving partial different equations (PDEs) applied to Earth-system models and to fluid dynamics models on high-end parallel architectures including heterogeneous architectures made of accelerators (NVIDIA GPU and Intel MIC). Relevant activities have also been conducted for optimising the management of the huge amounts of data produced by the climate models and introducing efficient approaches for the parallel I/O. He published more than 40 papers in refereed books, journals and conference proceedings on parallel, grid and cloud computing.

Silvia Mocavero is Scientist at the Advanced Scientific Computing (ASC) Division of the CMCC Foundation, where she leads the High-End Computing research group. She holds a PhD in Innovative Materials and Technologies awarded in 2006 from ISUFI at the University of Lecce (Italy). In 2010 she was visiting researcher at the Barcelona Supercomputing Centre (BSC) within the HPC-Europa2 initiative for the optimisation of the Biogeochemical Flux Model, coupled with NEMO. In 2013 she was visiting scientist at the Argonne National Laboratory (ANL) of Chicago for the porting, scalability analysis and improvement of the NEMO oceanic model on BlueGene/Q architecture. Her expertise concerns HPC, grid and cloud computing. Her skills include parallel programming on HPC systems and distributed environments, with deep experience on several programming models such as message passing, shared memory, many-threads programming with accelerators. Since 2011 she was exploring new issues related to the exascale computing. She works on the analysis and optimisation of climate models with a particular focus on NEMO as a member of the System Team and the HPC group of the NEMO Consortium.

Andrew Porter received his PhD in computational physics from the University of Cambridge, UK in 2000. Following a year working for a scientific consultancy, he joined the University of Manchester, UK. There he was the lead developer on the RealityGrid computational-steering framework, a component of the TeraGyroid project which won awards at both SC’03 and ISC 2004. Since 2007 he has worked as a computational scientist at the Science and Technology Facilities Council’s Hartree Centre, UK. He specialises in performance analysis and optimisation, particularly of Earth-system model components. Currently he is working on the code generation and optimisation system (PSyclone) used by the UK Met Office’s new LFRic atmosphere model.

Stephen M Pickles worked in commercial computing with ICL (Australia) before turning his hand to computational science. He gained his PhD in Lattice QCD from the University of Edinburgh in 1998, then joined the CSAR national supercomputing service at the University of Manchester where he soon became involved in grid computing. He was technical lead for the award-winning TeraGyroid project (SC’03 and ISC’04), Software Infrastructure Manager for the RealityGrid project, Technical Director of the UK National Grid Service, and Area Director for the Compute Area of the Open Grid Forum. He moved to STFC Daresbury Laboratory in 2008, where he specialised in the performance optimisation of parallel codes across a variety of scientific domains. He served briefly as leader of the Applications Performance Engineering Group before retiring from full-time employment in 2014. An Honorary Scientist at STFC Daresbury Laboratory since 2015, he still undertakes consultancy work in scientific computing, mainly during the colder months. During the warmer months, he opens Bidston Lighthouse, where he lives, to the public.

Mike Ashworth is Chief HPC Specialist, a senior management role in the Scientific Computing Department at STFC’s Daresbury Laboratory. He provides leadership in evaluation and exploitation of new technologies. He was one of the founders of the Gung-Ho project, which with the Met Office and NERC, is developing a new atmospheric dynamical core for weather prediction and climate applications. He also leads on many ocean model developments and has been a major contributor to NERC's Ocean Roadmap program. He has built up the Department's environmental modelling presence from nothing to a thriving activity including securing funding for projects on the POLCOMS, NEMO and ICOM ocean models, the UM and WRF atmospheric codes and a seismic wave propagation code for earthquake studies. He has played a major role in the UK National HPC Services establishing the HPCx Terascaling Team and leading the team in the last few years of the HPCx Service. For 14 years he led the Application Performance Engineering (APE) Group which focuses on the development and optimisation of large-scale applications for high-performance systems across a wide range of scientific disciplines, including evaluation and exploitation of novel architectures and the application of grid technologies.

Giovanni Aloisio is Full professor of Information Processing Systems at the Department of Innovation Engineering of the University of Salento, Lecce, Italy, where he is leading the HPC laboratory. Former director of the "Advanced Scientific Computing" (ASC) Division at the CMCC Foundation, he is now the Director of the CMCC Supercomputing Center and member of the CMCC Strategic Council. He is also member of the ENES (European Network for Earth System modelling) HPC Task Force. His expertise concerns HPC, grid and cloud computing and distributed data management. He has been the responsible for ENES of the EU-FP7 EESI-EESI2 (European Exascale Software Initiative) projects chairing the Working Group on Weather, Climate and solid Earth Sciences. He has also contributed to the IESP (International Exascale Software Project) exascale roadmap. He has been the chair of the European panel of experts on Weather, Climate and solid Earth Sciences (WCES) that has contributed to the PRACE strategic document "The Scientific Case for HPC in Europe 2015–2020". Presently, He is coordinating CMCC activities into several EU FP7 projects. At the University of Salento, he is the responsible for the EU H2020 EXDCI project, chairing the Working Group on WCES. He is the author of more than 100 papers in referred journals on HPC, grid computing and distributed data management.