# NEMORB's Fourier Filter and Distributed Matrix Transposition on Petaflop Systems

Tiago RIBEIRO and Matthieu HAEFELE
*Max-Planck-Institut für Plasmaphysik, EURATOM Association, 85748 Garching, Germany*

**Abstract.** The work presented refers to the calculation of two-dimensional Fourier transforms of distributed data, of which matrix transposition is a major ingredient. The motivation stems from a well-known parallel scalability bottleneck related to Fourier filtering within the global gyrokinetic NEMORB code, whose aim is to simulate plasma turbulence within a Tokamak fusion device. Since this pure MPI code is very HPC-resource demanding, with good scaling up to 65 536 tasks, such a bottleneck naturally impairs further parallel scalability. To overcome this limitation the filtering algorithm is modified. Firstly, exploring the Fourier transform Hermitian symmetry of the purely real-valued input data to discard the redundant Fourier modes. This not only reduces the amount of needed computing power but also the amount of communication between MPI tasks during the transpose phase. Secondly, several distributed transpose algorithms are tested and the high count of zeros yielded by the NEMORB's low pass filter is used to further reduce the inter-task data exchange by avoiding communicating zeros. Finally, the behavior of the transpose at the scale of a full petaflop system is explored. Execution time, MPI initialization and memory footprint can become critical on such large runs. Four implementation strategies have been tested and scaling results are shown on four petaflop systems, among which two belong to the PRACE Research Infrastructure.

**Keywords.** FFT, distributed matrix transpose, MPI, OpenMP, benchmark

## Introduction

ORB5 [1,2] in its current electromagnetic version NEMORB [3,4] is a global gyrokinetic particle-in-cell (PIC) code developed to study plasma turbulence within a toroidal fusion device called Tokamak. NEMORB is a highly scalable MPI parallel application using two main levels of domain decomposition, as depicted on Fig. 1. One level distributes the particles among so-called clones. Each clone has a three-dimensional (3D) grid representing the configuration space, which has a torus topology and is therefore naturally described by a radial coordinate ($s$), a poloidal ($\chi$) and a toroidal ($\phi$) angle, respectively. The charges and velocities of particles living in that clone are interpolated onto this grid. The contributions from all the clones are added up in order to compute the global electrostatic and electromagnetic fields. Finally the global fields are interpolated back onto the particles' positions in order to evaluate the pushing forces for each of them. The other decomposition level distributes one of the three dimensions of the spatial grid, namely, the toroidal angle $\phi$. The largest simulations performed use 1024 MPI tasks per clone and 64 clones, leading to an efficient scaling on 65 536 cores [5].

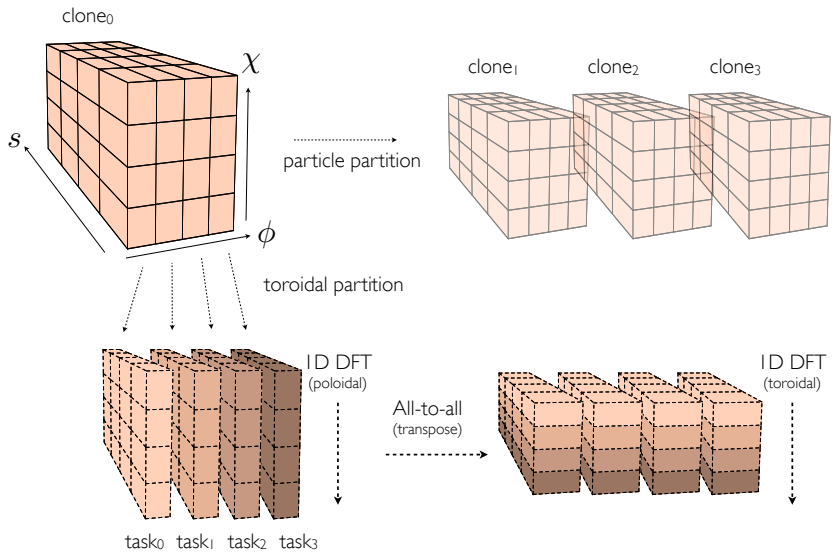


**Figure 1.** NEMORB's domain decomposition and two-dimensional discrete Fourier transform scheme.

During the computation of the fields, a set of angular degrees of freedom, or modes, that are disallowed by the nature of the Tokamak guiding magnetic field are filtered out. Failing to do so, necessarily wastes computing power, but worse, degrades the numerical signal-to-noise ratio, which is of central importance in a PIC code. In practice the filter implies the calculation at each iteration of two-dimensional (2D) Fourier transforms of the distributed 3D data, of which matrix transposes across tasks are a major ingredient. The discrete Fourier transforms (DFT) are performed on the poloidal and toroidal angles for each radial position. Especially for large sized-simulations, this represents a bottleneck that naturally impairs the code's performance and in practice renders plasma simulations for next generation Tokamaks (e.g. ITER and DEMO[1]) unfeasible. To overcome this limitation, NEMORB's filtering algorithm is modified and the corresponding steps are detailed in the remaining sections of this manuscript. Also noteworthy is that there are other codes of the EU fusion program that share the parallel numerical kernel with NEMORB (e.g. EUTERPE [6]) and can therefore directly profit from the results herein.

This paper is organized as follows. Sec. 1 discusses the basic profiling of the original NEMORB's 2D filter and introduces its main constituent blocks and domain decomposition. It lays the strategy to be followed in the remaining text. The Hermitian redundancy of the data and its exploitation are discussed in Sec. 2, where the first performance improvement results are shown. Secs. 3 and 4 discuss several transpose methods. The concept of using the high count of zeros resulting from NEMORB's Fourier filtering to perform only a partial transpose is explained and tested in Sec. 5. Sec. 6 departs from the strict framework of the NEMORB code to address the scalability of the distributed

[1] `http://ec.europa.eu/research/energy/euratom/fusion/iter/future/index_en.htm`

transpose up to several thousands of cores. At the end, a summary is provided in Sec. 7. Throughout this work, the following HPC systems have been used:

- Cray XE6 (90 112 cores): Edinburgh Parallel Comp. Centre (EPCC), U.K.
- IBM Blugene Q (458 752 cores): Jülich Supercomputing Center (JSC), Germany
- IBM Power6 575 (6 560 cores): Rechenzentrum Garching (RZG), Germany
- IBM x iDataPlex (147 456 cores): Leibniz-Rechenzentrum (LRZ), Germany
- Bull NovaScale R422-E2 system (8 640 cores): JSC, Germany
- Bull B510 blade system (70 560 cores): Computational Simulation Centre of the International Fusion Energy Research Centre (IFERC-CSC), Japan

## 1. Strategy

The basis functions for Fourier transforms are sines and cosines, which do not have a compact support (they are not spatially localized). Therefore, the computation of each Fourier coefficient requires contributions from all values in the original data. From a computational perspective, the local availability of the input data is of key importance to compute efficiently its DFT. The same obviously applies to $N$-dimensional DFTs that are obtained from a sequence of $N$ independent one-dimensional (1D) DFTs, one for each of the $N$ dimensions. Each of these needs its input data locally available on the task doing the computation.

NEMORB's 3D spatial grid is a discretization of a torus representing the Tokamak's plasma. It comprises the radial direction and the poloidal and toroidal angles $(s, \chi, \phi)$, discretized with $N_s$, $N_\chi$ and $N_\phi$ grid-nodes, respectively. Such domain is then decomposed into $N_{\text{tasks}}$ sub-domains over the toroidal direction and distributed across the same number of cores, typically as many as the toroidal grid-node count. This limits the number of parallel tasks to $N_{\text{tasks}} = N_\phi$. To overcome this, the domain is further decomposed into $N_{\text{C}}$ clones, each getting $1/N_{\text{C}}$ of the particles. The total number of parallel tasks becomes $N_{\text{tasks}} \times N_{\text{C}}$ [7] and this can scale up to 65 536 tasks [5]. However, the filtering operations in NEMORB occur independently within each clone and, for the purpose of this work, the analysis will be restricted to a single clone only, i.e. scalings up to $N_{\text{tasks}}$ cores only. For a given clone, the filter algorithm involves the calculation of DFTs of the two angles $\chi$ (poloidal) and $\phi$ (toroidal) for each radial position. As the toroidal dimension is distributed among several MPI tasks, the first direction to be Fourier transformed is the poloidal (local) one. Afterwards, a transpose in the angles is required to make the toroidal domain locally available on each MPI task. Only then can the evaluation of the toroidal DFT proceed efficiently, without communication penalties. Basic profiling of this part of NEMORB's kernel shows that, for the typical large simulation sizes under consideration here, the communication due to the transpose represents at least 2/3 of the filter cost. This motivates the strategy to improve the aforementioned algorithm, which naturally focuses on the transpose operation and can be split into three main parts. First, the exploitation of the Fourier transform symmetries, which reduces both the CPU and communication costs. Second, the optimization of the distributed transpose algorithm itself, which involves both tailored changes based on the knowledge of the filters involved, as well as attempts to use alternative methods. Finally, the last part attempts to investigate and improve the scalability of selected transpose methods on several thousands of tasks, beyond the current needs of NEMORB.

## 2. Real Fourier Transforms and the Hermitian Redundancy

NEMORB's 2D angular Fourier filter acts on the particle charges that are deposited on a spatial 3D grid representing the source term of the Poisson equation. Since these are purely real-valued quantities, the Hermitian symmetry of their poloidal Fourier spectra can be invoked to compute a real-to-complex (r2c) transform. The toroidal DFT acts upon the Hermitian reduced poloidal (complex) spectra, requiring a full complex-to-complex (c2c) transform, but only on the non-redundant modes that are roughly half of the original one, $N_\chi/2+1$. Moreover, not only does this reduce the amount of needed computations but also the amount of data to be exchanged through the transpose. Even though the XOR transpose method used here (refer to Secs. 3 and 4 for details) requires the array elements to be a power of two, which is not the case for the Hermitian reduced spectra, it is still possible to apply it avoiding zero-padding. We just need to invoke another symmetry of the poloidal spectrum, namely, that for even $N_\chi$, the zero ($F_0$) and Nyquist ($F_{N/2+1}$) Fourier modes are purely real. The real part of $F_{N/2+1}$ can be stored in the imaginary part of $F_0$ to reduce the number of poloidal elements by one, which is called half-complex packed format.

All together, there is an effective expected computing cost reduction of the whole 2D Fourier transform of roughly a factor of two relative to the original c2c method in NEMORB. Fig. 2 shows how the new r2c method strong scales with a grid representative of a typical ITER simulation. The runs have been performed on the two Bull systems dedicated to the fusion community (only Bull CSC is shown). The ratio between the elapsed times of the new Hermitian reduced 2D DFT to the original full-complex method is plotted. Barring scaling degradation issues for the 1D DFTs for higher numbers of cores, which invoke the FFTW library, the expected main result is confirmed. For the highest number of cores used, total speedups of about 1.5 and 2.0 were achieved on Bull JSC and Bull CSC, respectively. The plots further show that the algorithm is communication bound, since the total speedup curve follows closely the transpose corresponding curve. Therefore, the transpose constitutes the most important component of the algorithm, and it needs to be analysed in more detail.

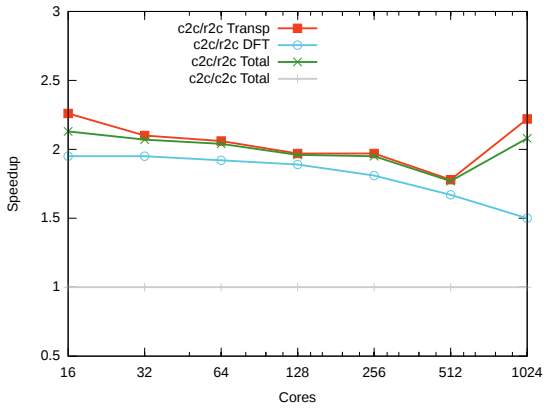


**Figure 2.** Strong scaling of a typical ITER-sized simulation ($N_s = 512$, $N_\chi = 2048$ and $N_\phi = 1024$). Speedup obtained by the Hermitian reduced 2D DFT on the Bull CSC machine.

## 3. NEMORB's Original Transpose Methods

The transpose methods originally available in NEMORB are analysed here. First let us recall that the communication pattern to perform a distributed matrix transpose is the so-called all-to-all pattern: each task sends a message to all the other tasks and each task receives a message from all the others. The `MPI_Alltoall` routine implements exactly this communication pattern and is therefore one option in NEMORB. Another possibility is to exchange explicitly data between all the tasks via point-to-point `MPI_Sendrecv` calls. All possible pairs of tasks are successively established thanks to a logical 'exclusive or' (XOR) condition [8].This will be referred as the XOR method hereafter and shall be explained in more detail in Sec. 4. From the memory representation point of view, the data that have to be sent are not contiguous, so neither `MPI_Alltoall` nor the XOR method can be used directly. The first option is to use additional send and receive buffers where the data is stored contiguously before and after the communication, respectively. The second option consists in using a dedicated MPI derived type to describe the reading and writing access layouts. These two options in combination with the two aforementioned communication methods yield four transpose methods, which have been tested. Their execution times are measured on the same strong scaling case of Sec. 2 and compared to the original Hermitian redundant XOR method (grey curve). The best case out of at least five runs is shown on Fig. 3 for Bull CSC, where each point on the graph represents the time it took to perform two thousand times the transpose, without doing any DFT in between. Actually, the transpose speedup scaling curves in Fig. 2 are obtained from the ratio of grey and red curves of Fig. 3. On both Bull systems, the best performing method is the XOR without MPI derived types. However, the same runs repeated on the IBM Power6 showed the best performing method to be the `MPI_Alltoall` with derived types, which reached a speedup of 2.4 on 1024 cores. These experiments show clearly that there is no absolute best performing transpose method. It is machine and/or MPI implementation dependent. As for the two fusion dedicated machines it is the XOR method without derived type that performs best, it takes the dominant role in what remains of this work.

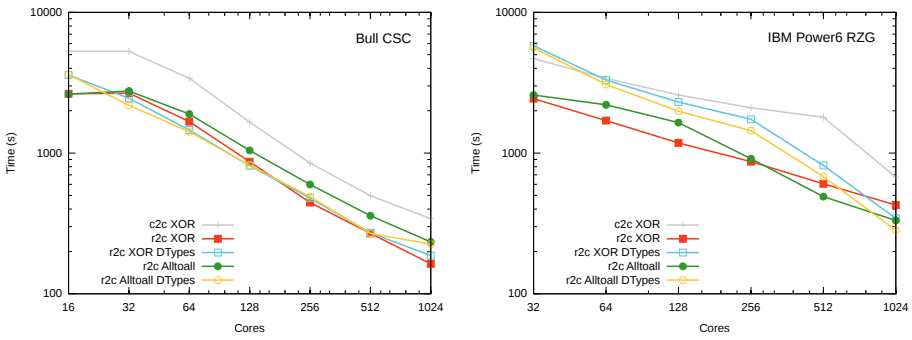


**Figure 3.** Strong scaling of a typical ITER-sized simulation ($N_s = 512$, $N_\chi = 2048$ and $N_\phi = 1024$). Execution time for different transpose methods on the Bull CSC (left) and IBM Power6 RZG (right) machines.

## 4. XOR vs. Alternative Methods

### *4.1. Transposes from FFTW and BLACS (via Intel MKL) Libraries*

The FFTW3.3 library and the Intel MKL implementation of the BLACS/ScaLAPACK library have also been tested. A simpler 2D version of the transpose sized ($N_\chi = 32768$, $N_\phi = 32768$) is considered and compared to the 2D version of the XOR method on 1024 tasks. FFTW's (resp. BLACS's) distributed transpose algorithm is three (resp. 58) times slower than the XOR method on the Bull CSC system. These results are somewhat surprising and suggest poor scaling (BLACS) Further investigations on this topic were not carried.

### *4.2. Transpose Using MPI Gather*

One can also achieve the distributed matrix transpose by using a sequence of consecutive `MPI_Gather` calls, one for each task. The underlying hierarchical trees implementation may perform better than point to point connections, but its blocking character prevents a task from starting a gather operation before the previous one is finished. This yields a serially blocking communication pattern. Assuming for simplicity that a single gather shows perfect weak scaling, the serial sequence of them done by all the cores increases the cost linearly. This phenomenon is clearly shown on Fig. 4 (left). Even though the efficiency of the hierarchical tree involved in the algorithm plays a role for small number of cores, its blocking nature prevents good scaling properties.

In turn, the XOR method exchanges data blocks according to a $N_{\text{tasks}} \times N_{\text{tasks}}$ pattern yield by the logical exclusive OR operation (`.XOR.`), as explained in the following lines and illustrated in Fig. 5. There, the map between the distributed data-set, represented on the left, and the corresponding XOR exchange pattern, shown on the right, is exemplified on a case parallelised over 8 MPI tasks. Recalling the spatial data domain decomposition illustrated in Fig. 1, one can easily identify the row $i$-indexes of the left matrix on Fig. 5 with the position of the data sub-blocks in the poloidal direction on Fig. 1. Similarly, the column $j$-indexes correspond to the toroidal indexes of the data sub-blocks, which also coincide with the MPI task ranks, since the domain is distributed in this direction. The

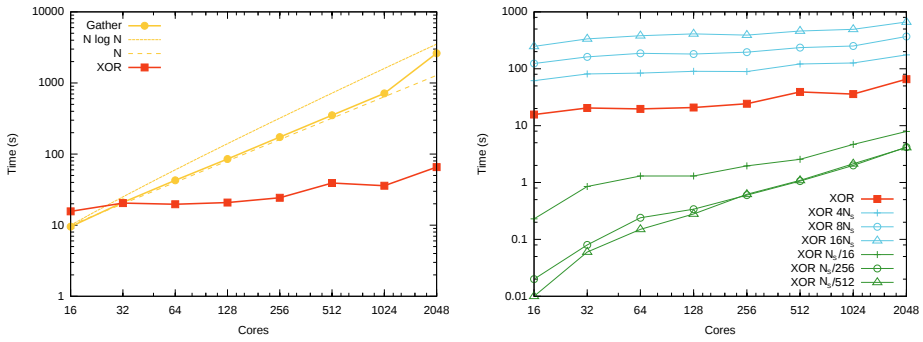


**Figure 4.** Execution times of two transpose methods for a weak scaling $N_\chi = 2048$, $N_s = 512$, $N_\phi = N_{\text{tasks}}$ (left). Execution times of the XOR method for a weak scaling using different message sizes (right). Measurements made on the Bull CSC machine.

matrix on the right has the same dimension and is similarly distributed across MPI tasks. Each of its elements holds the result of the $.\texttt{XOR}.(i,j)$ operation between its row and column indexes, i.e., between the corresponding sub-block poloidal index and the MPI core rank where it belongs to. They specify a one-to-one and onto map with the data-set matrix which yields the rules for its transposition via sub-block exchanges across MPI tasks. As this operation consists in switching rows and columns, and as each MPI task holds a whole column of data sub-blocks, the value of its rank ($j$) represents always the sub-block index ($i'$) at the destination. The rank of the destination task ($j'$) is specified by the value of the $(i,j)$-element of the right matrix, namely, $j' = .\texttt{XOR}.(i,j)$. To clarify this mechanism, the cases for the second and the third rows of the XOR matrix are realized using colors in Fig. 5. Let us take the red exchange pair on the lower example. For core rank $j = 0$, the value of row $i = 2$ of the XOR pattern is 2. This means that the data

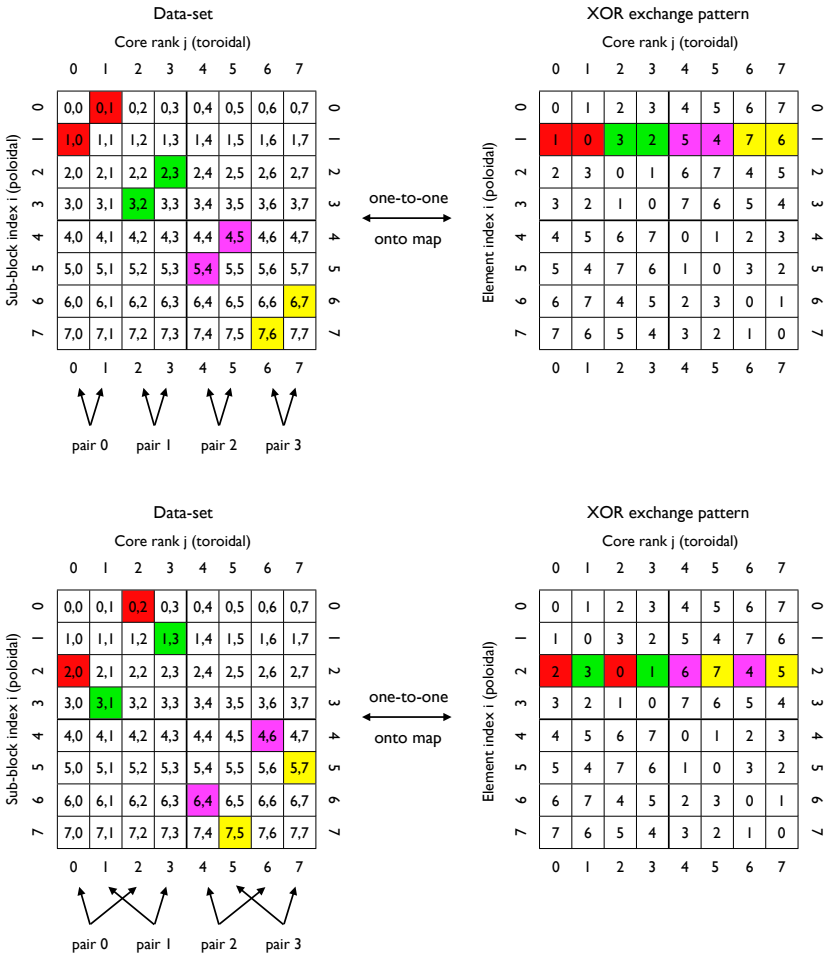


**Figure 5.** Illustration of the map between the data-set (left) and the XOR communication pattern (right) for $N_{\text{tasks}} = 8$. The correspondence between the second (upper) and third (lower) rows of the exchange pattern (right) and the actual data-set exchange pairs (left) is highlighted in color.

sub-block $i = 2$ of task rank $j = 0$ has to be sent to task rank $j' = 2$ at the sub-block position $i' = 0$. Conversely, the task rank $j = 2$ at position $i = 2$ has the XOR value of 0, which means that it has to send its data sub-block $i = 0$ to task rank $j' = 0$ and place it at the position $i' = 2$. Both together specify the red exchange pair on the left matrix. Each row of the exchange pattern matrix forms $N_{\text{tasks}}/2$ of such pairs that can be executed in parallel, provided $N_{\text{tasks}}$ is a power of two. Sequentially realizing all rows of the exchange pattern yields the full transpose.

Even though the (i) number of messages to pass increases quadratically with the number of tasks, (ii) their size decreases linearly and (iii) the amount of pairs available to exchange messages in parallel increases linearly. In the limit of large number of cores, the three factors balance and a flat curve is obtained, with the value set by the network bandwidth. Hence, unlike the previous method, this one is expected to have perfect weak scaling, as shown by the red curve in left plot of Fig. 4, which is mostly flat, at least until 256 cores. For higher numbers of cores the scaling starts to degrade due to network limitations. The plot on the right of Fig. 4 shows the same weak scaling for different message sizes. Both (i) the number of data pairs exchanged in parallel and (ii) the number of times such exchange is sequentially performed increase linearly with $N_{\text{tasks}}$. The former increases the probability of network congestion, whereas the latter implies a linear accumulation of latency. Moreover, since the size of the exchanged data blocks decreases accordingly, for smaller message sizes the communication times are dominated by the latency accumulation. This is shown by cases (green) with 256 and 512 times smaller messages than in nominal case (red on both plots). Those curves even become coincident beyond 128 cores, even though their sizes differ by a factor of two. Conversely, for large enough message sizes, the latency becomes sub-dominant, and if there were no other network limitations, a pure weak scaling (flat) line would be expected. However, the larger the message sizes are, the larger the slopes of the red curves become (note the logarithmic scales), which seems to indicate that there are too many messages being exchanged simultaneously for the system to cope with.

## 5. Optimization of NEMORB's Transpose

Besides having good scaling properties, the XOR method further allows to exploit the high count of zeros yielded by the NEMORB's low pass filter. The idea is to use the knowledge of the filter shape to inhibit parts of the XOR exchange pattern and alleviate the communication burden. This is possible because the exchange pattern is built "by hand" and, unlike with the methods relying on "canned" libraries, can be modified.

The basic filter used in NEMORB is a diagonal one, which retains only a relatively narrow band of poloidal modes around each toroidal mode in the system. It mixes both angular dimensions in a non-separable form and can only be applied after the 2D DFT is calculated, not before. On the other hand, a square filter is also applied to NEMORB's data to ensure a lower Nyquist frequency cut. Unlike before, it is possible to separate its angular projections. Hence, we can apply the filter's poloidal projection to the poloidal spectrum before it is exchanged across cores, and therefore avoid communicating the resulting zeros. Fig. 6 on the left illustrates these ideas on a Hermitian reduced case with $N_\chi = 16$, distributed over $N_{\text{tasks}} = 8$ cores, the same as in Fig. 5. The filtered-out interval corresponds to the lower part of matrix, coloured in magenta and yellow. The white

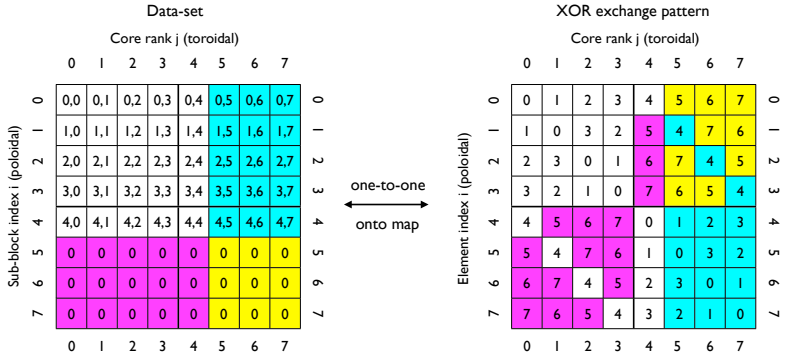


**Figure 6.** Effect of "four-node per mode" poloidal low-pass filter on the map between the data-set and the communication pattern. The yellow area shows the part of the communication that is inhibited by the filter.

regions remain unaffected. The lower right corner area (yellow) requires no communication whatsoever and inhibiting the corresponding part of the XOR exchange pattern effectively reduces the number of messages paths. The right side of Fig. 6 illustrates how this is done by using the colours to provide a pictorial view of the map between communication pattern and data-set, in the same fashion of Fig. 5. In comparison to the example there, of the eight exchange pairs shown there, this causes the two coloured in yellow to be inhibited. Obviously, the bigger $N_{\text{tasks}}$ is, the more pairs are affected. Compared to the results shown in Sec. 2, an improvement in performance of about $5-10\%$ is achieved, for the ITER grid-count. This improvement is likely due to the network congestion reduction as the number of exchanging pairs is reduced. As a result, the speedup factor increases from 2.0 (resp. 1.5) to about 2.2 (resp. 1.7) on Bull CSC (resp. JSC) system.

## 6. Scalability on Several Thousand Tasks

Leaving the strict framework of NEMORB's optimization, we use our algorithm testing suite to explore the behavior of a distributed matrix transpose at the scale of a full petaflop system. For NEMORB, this would mean running one single clone on 64k cores. Even if this is clearly not the case nowadays, as one clone runs typically on 1024 cores maximum, this question might arise in the future. We decided to take the user perspective to lead this study in order to gain practical information on current machine usage. All the results presented here are obtained with the XOR method without derived types, except where explicitly noted. As a reference case, let us consider the pure MPI implementation, called "flat" henceforth. In this context, an explicit all-to-all communication pattern using point to point connections implies a full connected set of MPI tasks. As the number of tasks increases, the number of inter-task connections to establish increases with $O(N_{\text{tasks}})^2$. This leads to three potential pitfalls. First, such large number of connections may increase the MPI memory footprint, leaving less memory to the application. Second, the cost of initializing the MPI environment may increase because creating the MPI processes and their inter-connections may take more time. Finally, the transpose requires the exchange of $O(N_{\text{tasks}})^2$ messages, so the execution time may also be impacted.

The experimental setup consists of a test case performing ten 2D matrix transposes successively. Weak scalings going up to 64k cores are performed on four architectures,

but CPU budget constraints imply that each run is performed only once. Nevertheless, every transpose is measured separately on each task, and the average, maximum and minimum are considered. As inter-tasks connections are actually established during the first MPI communication, the initialization time $t_{\text{init}}$ is defined as follows:

$$t_{\text{init}} = t_{\text{MPI\_Init}} + t_1 - t_{\text{exec}} \qquad \text{and} \qquad t_{\text{exec}} = \langle t_{5-10} \rangle = \sum_{i=6}^{10} t_i/5,$$

where $t_{\text{MPI\_Init}}$ is the duration of the call to `MPI_Init`, $t_1$ is the execution time of the first transpose and $t_{\text{exec}}$ the average execution time of the last five ones.

The implementation executes successfully on all four architectures used for our benchmark. The initialization time remains reasonable for all the machines except for the Bull CSC one. There, it exhibits a prohibitive cost, which should hopefully be fixed in the next Bull MPI release. Conversely, the initialization time is hardly measurable ($<$ 0.1 s) for the IBM BGQ. In between, Cray and IBM iDataPlex require 200 s to do the same on 64k and 32k cores, respectively. In Fig. 7, one can observe that the memory consumed by MPI on each task remains also reasonable for all machines. The amount of memory is neglectable for the IBM BGQ and oscillates between 40 and 110 MB for the IBM iDataPlex system. For the Bull CSC and the Cray systems, the MPI memory footprint presents larger variations. It is dominated by the message size for small numbers of cores and by the large number of connections for large numbers of cores. On the other hand, execution time of each transpose $t_{\text{exec}}$ can be costly on some systems. Bull CSC system performs very well by keeping it below 2.5 s up to 64k cores. For IBM iDataPlex, this timing stays below 3 s as long as a single island (8k cores) is used. Performance degrades substantially when using more islands. Both the IBM BGQ and Cray systems exhibit poor performance in this context, but we will see later that in these cases other transpose methods behave better.

On the way to exascale, the foreseen programming model remains an open question. Up to current petaflop systems, pure MPI implementations are still relevant for such a distributed matrix transpose problem. The question is now how far will such a pure MPI approach hold and what kind of alternative implementations are available that do not imply large software engineering overheads. Indeed, software engineering aspects matter a

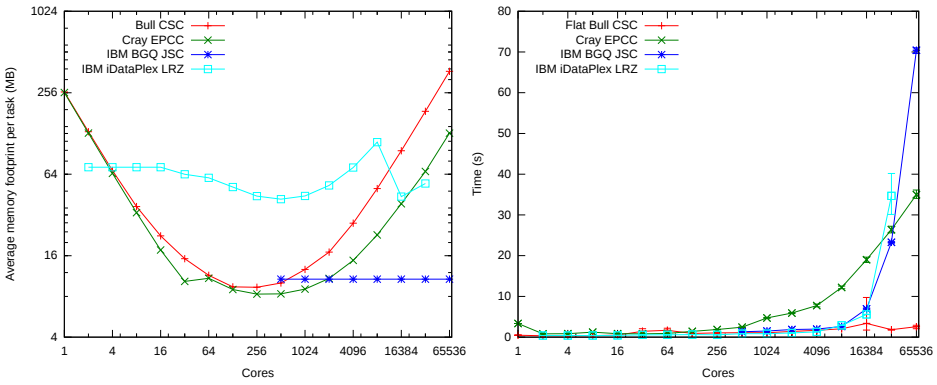


**Figure 7.** MPI memory consumption and single transpose execution time ($t_{\text{exec}}$) obtained on different systems for a weak scaling of the "flat" implementation with constant 128 MB sub-matrix data per core.

lot when one has to deal with large existing codes. Three alternative implementations are tested in this context. In order for the results to be comparable, the amount of data per node has been kept constant with respect to the "flat" implementation. First, the "hybrid" MPI/OpenMP approach enables to reduce the total number of tasks by having one MPI task per node instead of one per core. Second, the "shared memory" approach keeps one MPI task per core but, thanks to MPI communicators, only one task per node communicates with the other tasks outside the node. Inside the node, the different tasks share the data thanks to a shared memory segment. Third, the "two communication levels" keeps also one MPI task per core and only one communicating task per node, as in the previous approach. However, within a node, the data is gathered on the communicating task and then scattered back once the transpose is performed. The communicating task requires an additional memory buffer, thus increasing the memory consumption. From the software engineering point of view, the first two alternatives present large overheads as they imply modifying every loop nest that accesses the data, whereas the third one only changes the communication mechanism.

Whether or not these implementations will be mandatory on next generation machines is still an open question. What can be answered already is that they perform pretty well on nowadays Petascale systems. Compared to the "flat" implementation, the initialization time is reduced by an order of magnitude for Cray and IBM iDataPlex, even more for the Bull CSC system. The MPI memory consumption is also reduced by the same amount on all systems when the "hybrid" implementation is used. However, "shared memory" and "two communication levels" implementations exhibit similar MPI memory consumptions or even larger ones, which for the former is not yet fully understood.

From the execution time perspective, the situation is different on each system. The corresponding best results are shown on Fig. 8. For Bull CSC, the fastest method remains the "flat" implementation. The others do not perform badly but they are simply slower by 1 s approximately, which nevertheless represents a 50% degradation. For Cray, the picture is roughly the same except that the performance drops for large numbers of cores even for the alternative implementations. This situation was not expected and is still under study at EPCC. The actual best performance on the Cray system was obtained by

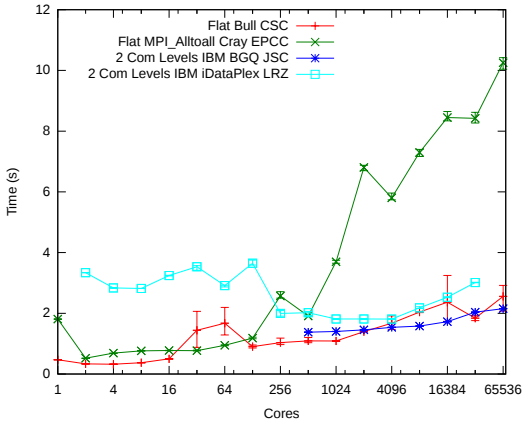


**Figure 8.** Best $t_{\text{exec}}$ among all implementations obtained for the same weak scaling of Fig. 7.

using the `MPI_Alltoall` routine instead of the XOR method. For the IBM iDataPlex, all three alternatives present very similar performance and do not degrade as much as the "flat" run on four islands. Finally, the IBM BGQ shows very different behavior for all these alternatives, the best one being by far the "two communications levels". We should also mention that the "shared memory" implementation could not be run yet on the IBM BGQ. We are still collaborating with IBM to solve this issue.

## 7. Conclusions

The Fourier filtering in NEMORB has been successfully improved by exploiting both the data Hermitian symmetry and the filter shapes to alleviate the communication burden. A speedup of two compared to the original implementation has been achieved. The total speedup of NEMORB is not yet available, and shall be the subject of further publication. Coming MPI 3 implementations with non blocking collective calls may also help to further improve this NEMORB's bottleneck by overlapping the DFT computations with the distributed matrix transposes.

On the way to exascale computers, four matrix transpose implementations have been evaluated on four different petaflop systems. The current pure MPI implementation is not yet a prohibitive limitation on nowadays supercomputers. However, how long this will hold is not clear. Further, if the hybrid MPI/OpenMP implementation showed in our study substantial improvements in MPI initialization time and memory footprint, it also never exhibited the best performance in execution time for the matrix transpose.

## Acknowledgements

The authors would like to thank R. Hatzky and T.M. Tran for useful discussions. A part of this work was carried out using the HELIOS supercomputer system at the Computational Simulation Centre of the International Fusion Energy Research Centre (IFERC-CSC), Aomori, Japan, under the Broader Approach collaboration between Euratom and Japan, implemented by Fusion for Energy and JAEA. The authors would also like to thank the other computing centers, EPCC, JSC and LRZ, for the access granted to their systems.

## References

[1] Tran, T. M. *et al.* Global gyrokinetic simulation of ion-temperature-gradient driven instabilities. In *Th. Fusion Plasmas*, Proc. of the International Workshop, Varenna, 45–49 (Ed. Compositori, Bologna, 1999).

[2] Jolliet, S. *et al.* A global collisionless PIC code in magnetic coordinates. *Computer Physics Communications* **177**, 409 – 425 (2007).

[3] Bottino, A. *et al.* Global nonlinear electromagnetic simulations of tokamak turbulence. *Plasma Science, IEEE Transactions on* **38**, 2129 –2135 (2010).

[4] Bottino, A. *et al.* Global simulations of tokamak microturbulence: finite-$\beta$ effects and collisions. *Plasma Physics and Controlled Fusion* **53**, 124027 (2011).

[5] Bottino, A. IFERC-CSC lighthouse project ORB5MULTI, final report. Private Comm. (2012).

[6] Jost, G., Tran, T. M., Cooper, W. A., Villard, L. & Appert, K. Global linear gyrokinetic simulations in quasi-symmetric configurations. *Physics of Plasmas* **8**, 3321–3333 (2001).

[7] Hatzky, R. Domain cloning for a particle-in-cell (PIC) code on a cluster of symmetric-multiprocessor (SMP) computers. *Parallel Computing* **32**, 325 – 330 (2006).

[8] Tran, T. M. Private Communication (1998).