applied sciences

MDPI

Article

Fast and Accurate Solution of Integral Formulations of Large MQS Problems Based on Hybrid OpenMP–MPI Parallelization

Salvatore Ventre [1,2], Francesca Cau [3], Andrea Chiariello [2,4], Gaspare Giovinco [5], Antonio Maffucci [1,2,*] and Fabio Villone [6]

[1] Department of Electrical and Information Engineering, University of Cassino and Southern Lazio, 03043 Cassino, Italy; ventre@unicas.it
[2] C.R.E.A.T.E. Consortium, 80125 Naples, Italy; andrea.chiariello@unicampania.it
[3] Fusion for Energy (F4E), 08019 Barcelona, Spain; francesca.cau@f4e.europa.eu
[4] Department of Engineering, University of Campania "Luigi Vanvitelli", 81031 Aversa, Italy
[5] Department of Civil and Mechanical Engineering, University of Cassino and Southern Lazio, 03043 Cassino, Italy; giovinco@unicas.it
[6] Department of Electrical Engineering and Information Technology, University of Naples Federico II, 80125 Napoli, Italy; fabio.villone@unina.it
* Correspondence: maffucci@unicas.it

Featured Application: Optimized parallel solution of large magneto-quasi-static problems.

Abstract: This paper proposes an optimal strategy to parallelize the solution of large 3D magneto-quasi-static (MQS) problems, by combining the MPI and OpenMP approaches. The studied numerical problem comes from a weak-form integral formulation of a MQS problem and is finally cast in terms of a large linear system to be solved by means of a direct method. For this purpose, two main tasks are identified: the assembly and the inversion of the matrices. The paper focuses on the optimization of the resources required for assembling the matrices, by exploiting the feature of a hybrid OpenMP–MPI approach. Specifically, the job is shared between clusters of nodes in parallel by adopting an OpenMP paradigm at the node level and a MPI one at the process level between nodes. Compared with other solutions, such as pure MPI, this hybrid parallelization optimizes the available resources, with respect to the speed, allocated memory, and the communication between nodes. These advantages are clearly observed in the case studies analyzed in this paper, coming from the study of large plasma fusion machines, such as the fusion reactor ITER. Indeed, the MQS problems associated with such applications are characterized by a huge computational cost that requires parallel computing approaches.

Keywords: eddy currents; fusion technology; integral equations; magneto-quasi-statics; MPI; OpenMP; parallel computing

Citation: Ventre, S.; Cau, F.; Chiariello, A.; Giovinco, G.; Maffucci, A.; Villone, F. Fast and Accurate Solution of Integral Formulations of Large MQS Problems Based on Hybrid OpenMP–MPI Parallelization. *Appl. Sci.* **2022**, *12*, 627. https://doi.org/10.3390/app12020627

Academic Editors: David Pardo and Luis E. García-Castillo

Received: 24 December 2021
Accepted: 6 January 2022
Published: 10 January 2022

1. Introduction

Low-frequency electromagnetic problems in the so-called magneto-quasi-static (MQS) limit (such as, for instance, eddy current problems) can be conveniently solved through numerical models coming from integral formulations, with the advantage of limiting the meshing to the conducting regions only. For real-world applications, however, the final formulation likely leads to large linear systems of equations, and suitable strategies are required to lower the computational cost of the numerical solution, such as those based on fast Fourier transform [1], fast multipole method [2], and H-matrix [3]. However, there are cases of interest where the dimensions are not so large to require the use of such techniques. For such cases, a direct solution method is preferable, since it provides intrinsic robustness and accuracy, with a modest increase in computational cost.

The final goal of this work is the optimization of the solution of integral formulations of eddy current problems solved with direct methods, in applications where long transients must be studied, and where an effective pre-conditioning is not possible. The applications targeted here come from the world of the nuclear fusion machines, and specifically refer to the evaluation of the current density induced in the conducting structures as a consequence of an electromagnetic transient associated with plasma disruptive events. The proposed optimization is based on the parallelization of the tasks associated with the numerical implementation of the integral formulation, based on a hybrid OpenMP–MPI approach.

The classical approach in parallel computing was based on the assumption that the systems were homogeneous, with each single node sharing the same characteristics: in this case, the computation time required by each node would be the same, if the same tasks were assigned to each single node. The present approach is based on the use of computational resources with high performance, made by cluster multicore systems [4], where variability and heterogeneity between the cores and between nodes introduces major issues, such as load imbalance, that strongly affect the effectiveness of parallel computation [5]. Optimizing the resources is of a primary importance for an efficient use of supercomputers, that are usually shared among several users. Traditional load-balancing techniques, such as those based on global re-partitioning of the threads, require a priori estimations on the time needed for completing each single task or, alternatively, require a dynamical reallocation of the loads based on a real-time check of the status of the nodes [6].

More recently, this problem has been addressed by using hybrid parallel programming paradigms, like those that adopt both a parametrized communication paradigm, such as the message passing interface, MPI [7], and an application programming interface, such as OpenMP [8,9]. Such a hybrid programming approach has been used in several fields, such as computational fluid dynamics [10,11], chemical simulations [12,13], geotechnics [14], high efficiency video coding [15], and electromagnetic simulations [16–18], due to its very interesting performance [19]. As pointed out in [19,20], the advantages of using a hybrid OpenMP–MPI programming approach are many, as follows: (i) suitability to the architecture of modern supercomputers, with interconnected shared memory nodes; (ii) improved scalability; (iii) optimization of the total consumed memory; (iv) reduction in the number of MPI processes. On the other hand, a few disadvantages can be observed, as follows: (i) complexity of the implementation; (ii) total gains in performance. These two issues require special care in the implementation of a hybrid approach, to handle the added extra costs to the existing code.

The main contribution of this paper is that of defining and assessing the implementation of such an approach when solving the integral formulation of eddy current problems by means of a Galerkin-based finite elements method, such as CARIDDI code, which is widely used in the fusion community [21]. The proposed solution fully exploits the main advantages of both paradigms: the MPI approach is more efficient in the process-level parallelization, whereas the OpenMP approach optimizes the lower level parallelization. The performance improvement is not only related to memory saving, but also to the reduction in the communication [22], with a consequent dramatic lowering of the latency time for the job to go in running state. The effectiveness of the proposed approach is witnessed by the obtained speed-up values for the analyzed real-world applications, about $\times 30$ and $\times 50$.

The paper is organized as it follows: Section 2 provides a short description of the numerical formulation of the MQS problem, and the tasks associated with building and solving the final numerical system are discussed. In Section 3, two different approaches for parallelizing the assembly of the system's matrices are presented and compared: a pure MPI approach and the proposed hybrid OpenMP–MPI one. In Section 4, first a benchmark test to validate the hybrid approach is provided. Then, the analysis of case studies referred to large nuclear fusion machines is carried out, comparing the computational performance of pure MPI and OpenMP–MPI approaches.

2. Numerical Formulation of the Magneto-Quasi-Static Problem

An electromagnetic problem in the so-called magneto-quasi-static problem (MQS) limit is studied here; namely, a low frequency eddy currents problem in non-magnetic conductors: these are typical conditions encountered in the applications analyzed in this paper, that are related to the plasma fusion machines, such as the fusion reactor ITER [23]. The mathematical model may be cast in integral form by introducing a vector potential, **A**, and a scalar one, ϕ, to define the electrical field, **E**, and the magnetic one, **B**, as follows:

$$\mathbf{E} = -\frac{\partial \mathbf{A}}{\partial t} - \nabla\phi, \quad \nabla \times \mathbf{A} = \mathbf{B}, \ (\text{Coulomb gauge } \nabla \cdot \mathbf{A} = 0) \tag{1}$$

In this way, Faraday's law is implicitly imposed. The vector potential is calculated from the current density, **J**, flowing inside the conductors' region, V_C, and from the density, $\mathbf{J}_S$, of the currents located outside that region, as follows:

$$\mathbf{A}(\mathbf{r},t) = \frac{\mu_0}{4\pi}\int_{V_C}\frac{\mathbf{J}(\mathbf{r}',t)}{|\mathbf{r}-\mathbf{r}'|}dV' + \mathbf{A}_s(\mathbf{r},t), \ \ \mathbf{A}_s(\mathbf{r},t) = \frac{\mu_0}{4\pi}\int_{R^3-V_C}\frac{\mathbf{J}(\mathbf{r}',t)}{|\mathbf{r}-\mathbf{r}'|}dV') \tag{2}$$

The integral formulation of the problem comes from the imposition of the constitutive equation $\mathbf{J} = \sigma\mathbf{E}$ in the ohmic conductors by means of the weighted residual approach, as follows:

$$\int_{V_C}\mathbf{W}\cdot\left(\sigma^{-1}\mathbf{J}-\mathbf{E}\right)dV = 0, \quad \forall\mathbf{W}\in S \tag{3}$$

The subspace S contains the functions solenoidal in V_C, satisfying the interface conditions for the current at the boundary: $S = \left\{\mathbf{J}\in\mathbf{L}^2_{div},\ \nabla\cdot\mathbf{J} = 0\ in\ V_C,\ \mathbf{J}\cdot\hat{n} = 0\ on\ \partial V_C\right\}$. By replacing (1) and (2) in (3), the final weak form is obtained, as follows:

$$\int_{V_C}\mathbf{W}\cdot\left\{\sigma^{-1}\mathbf{J} + \frac{\partial}{\partial t}\left[\frac{\mu_0}{4\pi}\int_{V_C}\frac{\mathbf{J}(\mathbf{r}',t)}{|\mathbf{r}-\mathbf{r}'|}dV' + \frac{\partial\mathbf{A}_s}{\partial t}\right]\right\}dV = 0, \quad \forall\mathbf{W},\ \mathbf{J}\in S \tag{4}$$

The above formulation is implemented in the code CARIDDI [21], that is here adopted to perform the numerical analysis of the considered case studies. The numerical model implemented in CARIDDI solves Equation (4) by applying Galerkin's method, starting from the following decomposition in edge elements, $\mathbf{N}_k$, of the unknown current density, as follows:

$$\mathbf{J} = \sum_k I_k\nabla\times\mathbf{N}_k \tag{5}$$

By using (5) in (4), under the assumption of time-harmonic signals, the vector, **I**, of the unknown coefficients, I_k, can be calculated by solving the following linear system:

$$\left(\boldsymbol{R} + \boldsymbol{L}\frac{d}{dt}\right)\cdot\mathbf{I} = \mathbf{Z}\cdot\mathbf{I} = \mathbf{V}_0 \tag{6}$$

The resistance $\boldsymbol{R}$ and inductance $\boldsymbol{L}$ matrices in (6) are calculated by means of the following integrals in the conducting regions, V_C:

$$R_{\mathrm{ij}} = \int_{V_C}\nabla\times\boldsymbol{N_i}(\mathbf{r})\cdot\sigma^{-1}\nabla\times\boldsymbol{N_j}(\mathbf{r})d\boldsymbol{r} \tag{7}$$

$$L_{\mathrm{ij}} = \frac{\mu_0}{4\pi}\int\limits_{V_C}\int_{V_C}\frac{\nabla\times\boldsymbol{N_i}(\mathbf{r})\cdot\nabla\times\boldsymbol{N_j}(\mathbf{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|}d\boldsymbol{r}d\boldsymbol{r}' \tag{8}$$

whereas $\mathbf{V}_0$ is a known vector that is computed from the external imposed sources, as follows:

$$V_{0,k} = -\frac{d}{dt}\int_{V_C}\nabla\times\boldsymbol{N}_k(\mathbf{r})\cdot\boldsymbol{A}_s(\mathbf{r},\mathrm{t})d\boldsymbol{r} \tag{9}$$

Once the numerical problem (6) is solved, the flux density **B** is obtained at any given position **r** as $\boldsymbol{B} = \boldsymbol{Q}\cdot\boldsymbol{I}$, where the generic entry of the matrix $\boldsymbol{Q}$ is defined as follows:

$$Q_{ki} = \frac{\mu_0}{4\pi}\int_{V_C} \frac{(\nabla x \boldsymbol{N}_k(\mathbf{r}')) \times (\mathbf{r_i} - \mathbf{r}')}{|\mathbf{r_i} - \mathbf{r}'|^3} d\mathbf{r}'. \quad (10)$$

The solution of the linear system (6) via a direct method requires a first step of matrix assembly, followed by a final step of matrix inversion. Specifically, the following four tasks may be identified:

(a) assembly of the known-terms vector, $\mathbf{V}_0$;
(b) assembly of the resistance and inductance matrices, **R** and **L**;
(c) assembly of the flux density matrix, $\boldsymbol{Q}$;
(d) inversion of the impedance matrix, $\boldsymbol{Z}$, defined as in (6), via factorization and back substitution.

In real-world applications such as those analyzed in this paper, related to fusion machines, the high computational burden requires the use of efficient parallel computing techniques, as discussed in the next section. In this paper, we focus on the parallelization of the assembly phase; whereas, the matrix inversion is here performed by using the best available routines, such as the Scalapack one [24]. The "inversion phase" (task (d)) is usually the most time consuming when using a direct solver, but it is worth noticing that in many practical cases also tasks (a)–(c) in the "assembly phase" can be very demanding of both CPU and RAM resources. Indeed, if we denote with N_{dof} the degrees of freedom of the numerical problem, the time required for the inversion (t_i) is proportional to N_{dof}^3, $t_i = k_i N_{dof}^3$; whereas, the assembly time (t_a) is proportional to N_{dof}^2, i.e., $t_a = k_a N_{dof}^2$. Following this consideration, one can be brought to affirm that the assembly time is lower than the inversion one. However, in many applications, the actual CPU-times for the two phases are comparable, since k_i depends only on the computational resources, whereas k_a depends on the accuracy. Indeed, there are some cases of practical interest among the MQS problems where the required accuracy imposes values of k_a that may lead to values of t_a comparable to, or even greater than, t_i. This happens, for instance, when the density of the electrical current varies rapidly in space and in time and consequently a high accuracy in the evaluation of the inductance matrix is required. Let us note that a higher accuracy may be achieved in two ways, as follows: (i) by refining the meshing, hence increasing N_{dof}; (ii) by using the same mesh (hence the same N_{dof}), but increasing the number of integration Gauss points, n. It is evident that solution (i) is not preferable when direct methods are used, given the cost of inversion. Let us then study the sensitivity of the assembly cost to increasing the values of n. The simulation times required for the inversion of (6) and for assembly of the matrix, **L**, are reported in Figure 1, as a function of N_{dof}. For a lower accuracy ($n = 1$), t_i is always greater than t_a, but for a better accuracy ($n > 1$), the assembly time is longer than the inversion one for $N_{dof} < N_{dof}^*$, where N_{dof}^* increases with n—it is 6500 for n = 270,000 for n = 3, and 351,000 for n = 4.

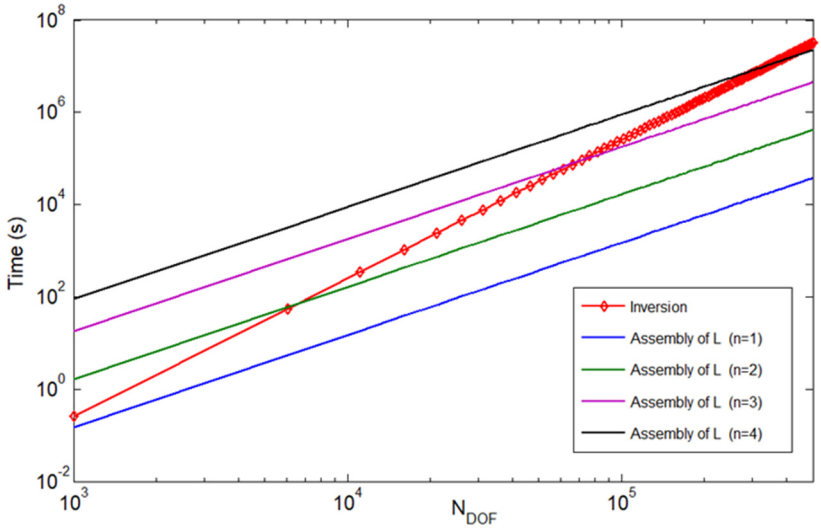

Figure 1. Benchmark case study: times required for the inversion of (6) and for the assembly of **L** versus the number of degrees of freedom, N_{dof}. The assembly times are parametrized to the number of Gauss points, n.

Another source of computational burden is the calculation of the matrix **L**, i.e., task (b): the double integration in (8) requires a proper handling of the singularities in the kernels, to retain the property of **L** to be positive definite. An adaptive procedure for integrating these singularities is given in [25].

3. Parallel Computing Based on a Hybrid OpenMP–MPI Approach

3.1. Parallelization Strategy Based on MPI Approach: Description and Limits

The numerical model of the MQS problem, described in the previous section, has been implemented in a parallel-computing scheme, based on a pure MPI approach, which is here briefly summarized (further details are given in [26]).

As for task (a), when assembling the known-terms vector, $\mathbf{V}_0$, the sources are the impressed currents, given in an external mesh (the sources mesh). The computation of the entries of $\mathbf{V}_0$ requires a double loop: the first on the mesh elements, the second one on the external mesh (as shown in Algorithm 1). Actually, $\mathbf{V}_0$ is a small size vector, and hence, although each MPI process allocates the whole vector in V0_loc, only a part of its memory is used. Then, a reduction operation is required in order to gather the final vector.

Algorithm 1 Evaluation of **V0**, pure MPI approach

```
// Initialization
Split Source Points between MPI processes
// Parallel pure MPI computation
for each MPI process do
    for each iel mesh element point do
        for each iel0 source mesh element do
            compute V0 contribution, between iel and iel0
        end for
    end for
end for
// All reduce of the local contribute on the global V0
mpi_allreduce(V0_loc,V0_global)
```

The assembly cost for task (b) is mainly related to the inductance matrix **L**, being **R** a sparse matrix [26]. The choice of the most suitable parallelization approach for computing the **L** matrix comes from a trade-off between the needs of optimizing the costs of computa-

tion, memory writing, and communications. For this purpose, two main strategies have been in the past investigated by the authors [26]: in the first one, the element by elements interactions needed to compute the integrals in (8) are distributed all over available MPI processes; in the second one, each process takes care of building a sub-block of the **L** matrix. Overall, numerical experiments in [26] show that the first approach proved to be the most effective one, given the characteristics of the transient electromagnetic problems of interest in the study of nuclear fusion machines. For this reason, we have adopted such an approach, that is here implemented by means of a hybrid OpenMP–MPI paradigm.

Looking at its definition in (8), the assembly of **L** requires two nested loops on the mesh elements, to compute the element–element interactions, as shown in Algorithm 2. To this end, a critical point is the need to accumulate and store these interactions in a correct way. Three dummies' memories are defined:

- M_{DMESH}, used to store the geometrical mesh information. These data have a dimension of the order of magnitude of the number of the mesh elements;
- M_{DL}, used to temporarily store the entries L_{ij} produced during the main loop of element–element interactions. This memory has the same dimension of M_{LOC}, that is the chunk memory required for matrix storage at each node;
- M_{DEE}, used to carry on the main loop of element–element interactions: such a memory is of the order of $N^2_{dof,el}$, being $N_{dof,el}$ the number of degrees of freedom per element.

Algorithm 2 Assembly of **L** matrix, pure MPI approach

```
Equally distribute element-element interactions among MPI processes
// Initialization
for each MPI process
    Allocate dummy memory MdMesh
    Allocate dummy memory MdEE
    Allocate dummy memory MdL
end for
Broadcast the geometrical information
// Parallel pure MPI computation
for each MPI process do
    for each element iel1 do
        for each element iel2 do
            Compute local iel1-iel2 interactions
            Accumulate local interactions in MdL
        end for
    end for
end for
//Final Communications Step
for each MPI process do
Allocate local memory Mloc
end for
for each MPI process do
    Send and receive the local matrices MdL
    Accumulate in Mloc
end for
deallocate(MdMesh, MdEE,MdL)
```

As for task (c), the assembly of the flux density matrix, **Q**, is obtained through a double loop, the first on the mesh elements, and the second one on the requested field points, as shown in Algorithm 3. In a pure MPI approach, the set of field points are equally distributed among MPI processes. Memory is then allocated in each MPI process.

Algorithm 3 Evaluation of **Q** m, pure MPI Approach

```
// Initialization
Equally distribute field points among MPI process
Broadcast the geometrical information
// Parallel pure MPI computation
for each MPI process do
    for each mesh element iel do
        for each field point ifp in the set belonging to current process do
            Compute Magnetic Field or Vector potential
            Accumulate the values
        end for
    end for
end for
//Final allreduce
mpi_allreduce(MagField,Vector potential)
```

As a general comment, we stress that with the pure MPI paradigm, due to the limited size of per node memory and to the need of each MPI process to allocate not only the memory required to store the local portion of the matrices but also the dummy memory required to carry on the computation, it may not be possible to launch a number of MPI process equal to the number of the physical cores available at the node. For the above reasons, many of these cores are forced to stay idle during the job execution because there is no more space to allocate. This fact limits the maximum achievable speed-up.

For example, to compute the **L** matrix, the mesh information should be available at each MPI process in the dummy memory *MdMesh* defined above: in practical applications, this memory likely reaches the dimensions of some GBs. In the following, we refer to the typical case of homogeneous supercomputing systems, made by nodes that are equal in terms of core number, memory, and operating frequency. In order to assess the performance, let us here define the following quantities:

- N_{MPI}—number of the MPI processes;
- N_N—number of the cluster nodes;
- M—total memory required to store the global matrix (for example, **L**);
- M_N—total memory available at any node;
- M_D—dummy memory per MPI process;
- M_{DT}—dummy memory per thread;
- M_{LOC}—memory available at each node (for all is needed at the node);
- M_{AV}—actual available memory at each node.

In the ideal case, the maximum computation speed-up would be equal to the total number of MPI processes, N_{MPI}. At each node, the number of running processes is N_{MPI}/N_N, and thus the available memory is given as $M_{LOC} = M/N_N$. Note that the available memory must be used not only the matrix storage (chunk memory) but for any other data required by the process. Unfortunately, the dummy memory, M_D, is replicated N_{MPI}/N_N times, due to the distributed memory approach adopted here. Therefore, the actual available memory at each node, M_{AV}, is reduced with respect to M_N, and must be larger than the requested chunk memory, M_{LOC}. These conditions are summarized in (11), that sets the limit for the pure MPI approach by imposing a maximum value to N_{MPI}:

$$M_{AV} = M_N - \frac{N_{MPI}}{N_N} M_D \geq M_{LOC} = \frac{M}{N_N} \tag{11}$$

3.2. Parallelization Strategy Based on Hybrid OpenMP–MPI Approach

A hybrid OpenMP–MPI approach is here proposed to overcome the abovementioned limits of the pure MPI approach. The underlying idea is to allocate in each node only one instance of the M_D memory, and then to use all the physical cores only for computation. This means that we must logically separate allocation from computation, that is forbidden

in a pure MPI paradigm, which maps one-to-one cores and processes. The solution could be provided by the use of the OpenMP environment directives: the node computation is broken down among the physical cores (threads) present in the node, that share the same memory on the node thanks to the shared memory approach implemented by OpenMP.

A hybrid OpenMP–MPI approach may be then implemented, based on the two following steps:

(1) as in the pure MPI paradigm, the overall computation is partitioned in MPI processes, limiting the number of the processes at node level (in the ideal case this number would be 1);
(2) the computational burden of each MPI process at node level is divided (again) in several threads, in accordance with the characteristics of the OpenMP paradigm.

In order to investigate the memory allocation scheme of the hybrid approach, we firstly recall that in OpenMP environment, each thread should allocate its local memory in order to face its own computation. Note that the dimension of the dummy memory per thread, M_{DT}, is much lower than M_D. In order to speed up the inter-thread computation and to fully gain from the modern CPU architecture, M_{DT} should match the local core cache size (*cache coherence*, e.g., [27]). Let us now define as N_T the number of threads per node: if we use one MPI process per node, and after considering that M_D is allocated only once, Condition (11) on the available memory for the pure MPI approach is replaced for the hybrid OpenMP–MPI one by the following:

$$M_{AV} = M_N - N_T M_{DT} - M_D \geq M_{LOC} = \frac{M}{N_N} \tag{12}$$

In view of comparing the two approaches, it is convenient to introduce the following speed-up parameters: $S_{MPI} = N_{MPI}$ (for pure MPI approach) and $S_H = N_T N_N$ (for hybrid OpenMP–MPI). They can be expressed from (11) and (12), as follows:

$$S_{MPI} \leq \frac{M_N}{M_D} N_N - \frac{M}{M_D}, \;\; S_H \leq \frac{M_H - M_D}{M_{DT}} N_N - \frac{M}{M_{DT}} \tag{13}$$

The above relations set the upper bounds for the two speed-up parameters, that linearly depend on the number of nodes, N_N: Figure 2 shows the corresponding straight lines, whose intersection occurs at:

$$N_N^* = \frac{(M_{DT} - M_D)M}{(M_{DT} - M_D)M_N + M_D^2} \tag{14}$$

where $N_N^* > 0$, since $M_N > M > M_D \geq M_{DT} \geq 0$. For $N_N > N_N^*$, it is always $S_H > S_{MPI}$ and the distance between the two bounds increases monotonically, as shown in Figure 2.

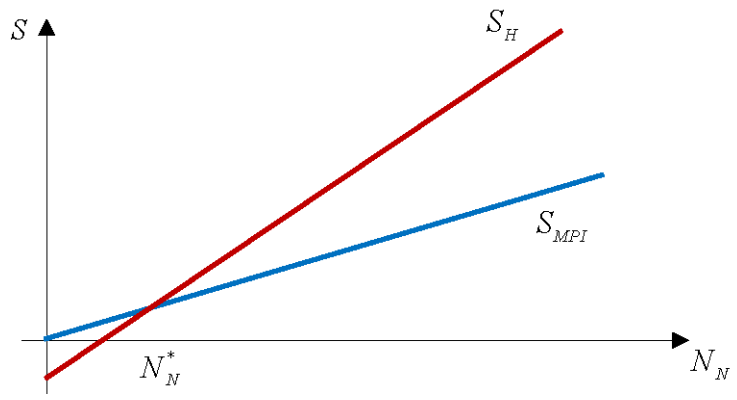

Figure 2. Upper bound for the speed-up parameter as a function of number of nodes N_N: pure MPI approach (S_{MPI}) and hybrid OpenMP–MPI approach (S_H).

Indeed, this result can be demonstrated by considering the ratio between the slopes of the straight lines—from (13), as follows:

$$\frac{\frac{M_N}{M_D}}{\frac{M_N - M_D}{M_{DT}}} = \frac{M_{DT}}{M_D}\frac{M_N}{M_N - M_D} < 1 \tag{15}$$

where the inequality holds since $M_{DT} < M_D$, and $M_D \ll M_N$, hence $M_N/(M_N - M_D) \approx 1$.

As clearly shown in Figure 2, this hybrid OpenMP–MPI approach provides two advantages over the pure MPI approach, as follows:

(i) resources saving—for a fixed speed-up S, the required number of nodes N_N is lower;
(ii) speed-up advantage—for fixed resources (N_N), the speed-up, S, is higher.

In addition, as pointed out in the Introduction, the approach also reduces the communication burden, because the parallel routines impose a fast communication between each thread in the node.

It is worth noticing that it is not trivial to accomplish OpenMP–MPI paradigm, since three critical issues arise, as follows:

(i) Need for local thread memory— the main loop is distributed among all available threads, each of them requesting its local memory to work properly. This memory is related to mesh element information (e.g., the curls of the elements, the shape functions, geometrical information, element local output, and so on). Anyway, it is usually very small (few GBs) and, in cases of practical interest, is much smaller than the required output global node memory.
(ii) Global matrix memory update—once the single thread made its own job, the thread local output should be accommodated into the global memory of the node. This is a non-trivial operation, and it is a bottleneck for the method, because the local update must be carried out by each thread in a sequential access, so to guarantee consistency. To this end, the CRITICAL OpenMP directive is used [6].
(iii) Global input memory access—this access is critical being shared between the threads. It may cause "cache missing".

Of course, all of the above issues cause a degradation of the performances; moreover, as we can see in Section 4, they do not significantly affect the obtained speed-up parameters.

Let us now describe the changes introduced by the proposed hybrid approach in the algorithms for assembling the matrices and vectors of the Numerical Problem (6). In building the known-terms vector $\mathbf{V}_0$, a hybrid implementation of the external loop (over the mesh elements) is handled by all threads. The local memory V0_loc is automatically obtained by the means of OpenMP reduction operations (Algorithm 4).

The new algorithm to be used to assembly the matrix **L** is described in Algorithm 5. In the MQS applications, such as those analysed in this paper, the fully populated matrix **L** is by far the largest quantity. The entries of this matrix are arranged in a 2D cyclic block size fashion, in view of using the Scalapack inversion routine [24,26]. To this end, the entries MdL are reorganized in M_{LOC} in the last step (final communication step). Once again, we stress the advantage of the hybrid approach over the pure MPI approach: since it is usually $N_N \approx N_{MPI}$, the memories MdMesh and MdL (by far the largest ones) are allocated only once in the node, instead of being allocated for each process in the pure MPI approach.

Finally, the **Q** matrix can be very large if the computation of the flux density is required in a huge number of points and/or in each element of the mesh. With the hybrid approach, the external element loop is distributed among the threads in the current MPI process, see Algorithm 6.

Algorithm 4 Evaluation of **V0**, hybrid OpenMP-MPI Approach

```
// Initialization
Split Source Point between MPI process
// Parallel pure MPI computation
for each MPI process do
    for each iel mesh element point do
        #pragma omp parallel for
        #pragma ompreduction(+:V0loc)
        for each iel0 source mesh element do
            compute V0_loc
        end for
    end for
end for
// Synchronization point
mpi barrier
// All reduce of the local contribute on the global V0
mpi_allreduce(V0_loc,V0_global)
```

Algorithm 5 Assembly of **L** matrix, hybrid OpenMP-MPI approach

```
Equally distribute element-element interactions among MPI processes
// Initialization
for each MPI process do
    Allocate dummy memory MdMesh
    Allocate dummy memory MdL
    Allocate dummy memory MdEE
end for
for each MPI process do
    Broadcast the geometrical information
end for
// Hybrid MPI openMP computation
for each MPI process do
    Declare MdEE as private for each thread
    #pragma omp parallel for
    for each element iel1 do
        for each element iel2 do
            Compute local iel1-iel2 interactions
            Compute the local interactions on a private dummy mem. MdEE
            #pragma omp critical
            Accumulate the local interactions on the shared mem. MdL
        end for
    end for
end for
for each MPI process
    Allocate local memory Mloc
end for
//Final Communications Step
for each MPI process do
    Send and receive the local matrices MdL
    Accumulate the contribute on Mloc
end for
deallocate(MdMesh, MdEE,MdL
```

Algorithm 6 Evaluation of **Q** matrix, hybrid OpenMP-MPI Approach

```
// Initialization
Equally, distribute field points among MPI process
Broadcast the geometrical information
// MPI OpenMP computation
for each MPI process do
    for each iel mesh element do
        #pragma omp parallel for
        for each ifp field point do
            Compute Magnetic Field or Vector potential
            Accumulate the values
        end for
    end for
end for
//Final allreduce
mpi_allreduce(MagField,Vector potential)
```

4. Case Studies and Discussion

The hybrid OpenMP–MPI approach presented in this paper has been implemented in two parallel computing systems with different characteristics: a proprietary system (SUNCUDA Cluster) and a supercomputing facility (MARCONI Cluster) [28]. The details of such systems are given in Table 1. Note that the proposed hybrid paradigm is general and can be applied to computing systems with different features.

Table 1. Features of the adopted parallel systems.

	SUNCUDA Cluster	MARCONI Cluster
Number of nodes	2	3216
Number of processors per node	2	2
Processor type	Intel Xeon E5-2690@2.9 GHz	Intel Xeon 8160@1.10 GHz
Number of cores per processor	8	48
RAM at each node	128 GB	192 GB

4.1. A Benchmark Case Study

A preliminary case, that is studied as a benchmark, refers to the simple MQS problem described in Figure 3, with a pancake coil that is inducing currents into a nearby conducting plate. The numerical model of the MQS problem is characterized by a mesh with 13,500 elements, 15,376 points, and 25,650 DoFs. Although the size is not so large to require a parallel computation approach, we analyse this case study with the only scope of comparing the performance of OpenMP only and OpenMP–MPI approaches.

The assembly of V0 and L (tasks (a) and (b) defined in Section 2) have been performed by using both OpenMP only and OpenMP–MPI approaches, and the results are provided in Table 2. Specifically, the speed-up obtained has been evaluated with respect to the use of 1 OpenMP thread and 1 MPI node, that is assumed as the reference. The computation times for the reference case are 8.93 s and 2693 s for tasks (a) and (b), respectively. Table 2 shows an almost linear increase in the speed-up values versus the number of threads, with a slight degradation obtained for task (b). The degradation comes from two major critical issues, as follows: (i) a larger thread memory is needed for evaluating integral (8), that is characterized by element–element interactions; (ii) the output of this calculation has to be stored at the node level in a matrix that is shared among all the threads. To guarantee consistency, the update of such a matrix is performed by a sequential access of the threads. The advantage of using the proposed hybrid OpenMP–MPI approach is the possibility to optimize the memory and to obtain the same speed-up for both the assembly tasks.

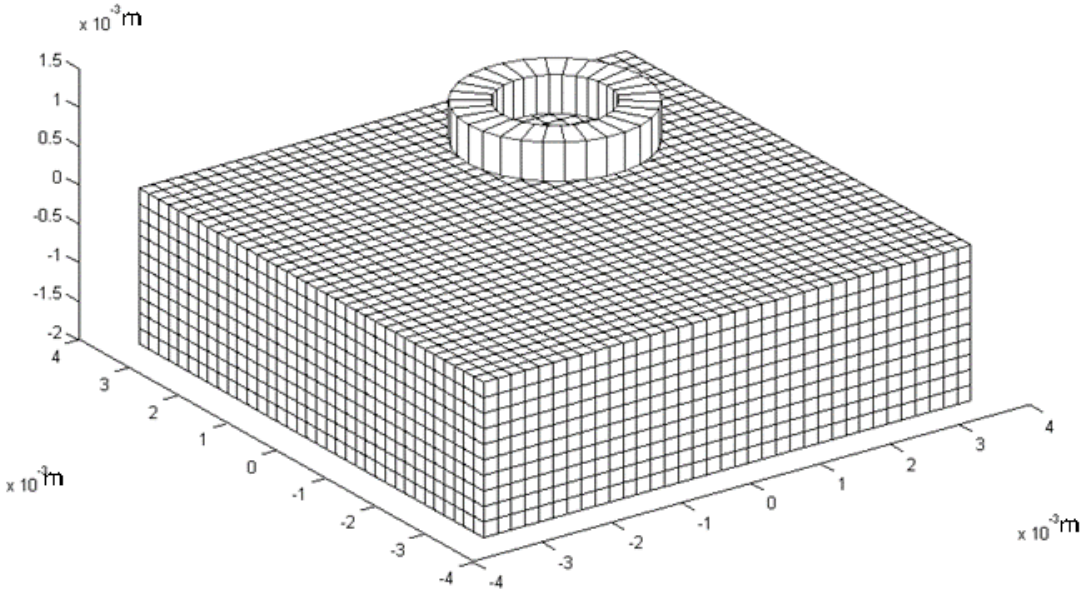

Figure 3. The benchmark case study: a pancake coil over a conducting plate.

Table 2. Benchmark case: speed-up values for tasks (a) and (b) with OpenMP–MPI on SUNCUDA Cluster.

MPI Tasks	OpenMP Threads	Speed-Up Values	
		Assembly of V_0	Assembly of L
1 (reference)	1 (reference)	1	1
1	4	4.0	3.4
1	8	7.4	6.7
4	1	3.7	3.8
4	4	14	15
4	8	26	27

4.2. Case Study 1: A Plasma Ring

Case study 1 refers to the study of the interaction of a plasma ring subjected to an asymmetric kink instability in the presence of a conducting vacuum vessel and a set of toroidal field coils, as shown in Figure 4. The kink is described by assuming the plasma as a single tilted and shifted filament, with respect to its axisymmetric equilibrium position. In Figure 4a, we show a sector of 90° of the total mesh, together with the surface enclosing the plasma rings necessary for integrating the Maxwell stress tensor. In Figure 4b, we show the time history of the x component of the resultant force due to the interaction between the plasma ring and the toroidal field, computed by means of the $\boldsymbol{j} \times \boldsymbol{B}$ expression and by using the Maxwell stress tensor. We notice that this method is very efficient in presence of a 3D plasma current distribution with time-varying shape and position. For this case, we adopted a mesh with 44,200 elements, 88,400 nodes, 12,600 points, and N_{dof} = 44,202. For eddy current problems in nuclear fusion machines, such as in this case study and in case study 2, discussed later on, a satisfactory estimation of the current density, $\boldsymbol{j}$, in the conductors, comes from the choice of linear basis functions and a number of Gauss points equal to 2, in the numerical computation of the integrals—see [21].

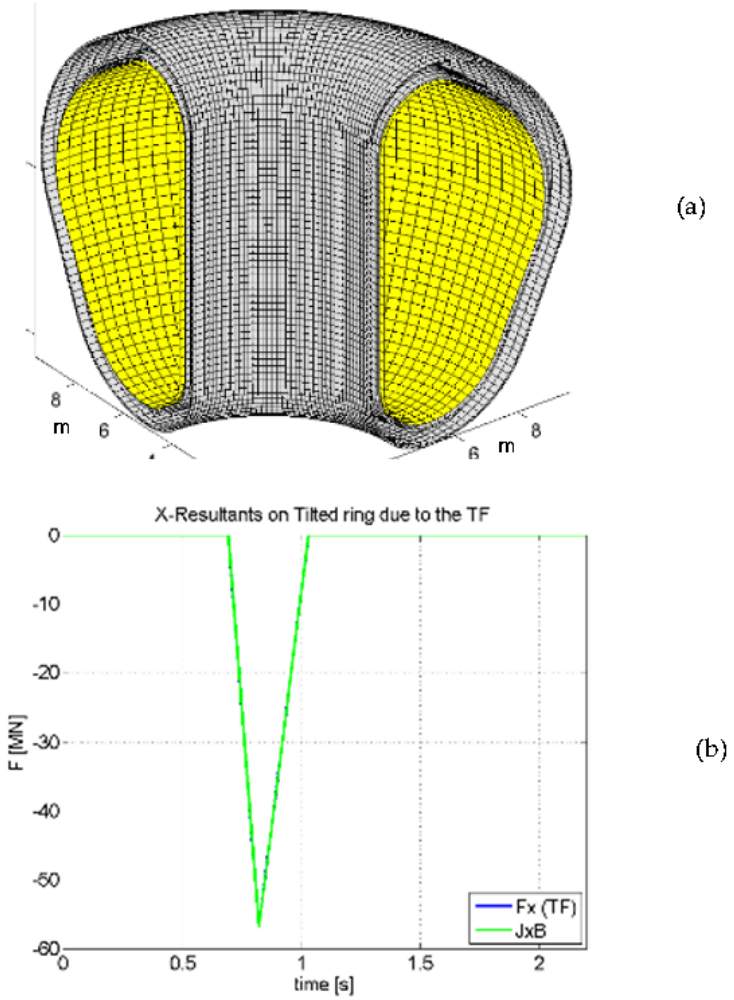

Figure 4. Case study 1: (**a**) one fourth of the double wall vacuum vessel and of the surface (in yellow) for the integration of the Maxwell stress tensor; (**b**) the time history of the x resultant on the tilted filament due to the toroidal field (TF).

For this case study, the computational effort for task (c) has been evaluated—evaluation of the flux density matrix $\boldsymbol{Q}$. Here, the reference time is given by the solution evaluated with 10 nodes, 2 MPI tasks per node, and 1 OpenMP thread: such a time is 1932 s on the Marconi cluster. Table 3 shows the speed-up values as functions of the total nodes, of the OpenMP threads and of the MPI tasks. The speed-up value depends almost linearly by the number of MPI tasks and OpenMP threads. It is noteworthy that this performance is obtained with a proper handling of the memory burden, since the memory is equally partitioned between the nodes involved in the computation. From Table 3, we can see that using 40 nodes and 1 MPI task per node provides a lower speed-up with respect to using 20 nodes and 2 MPI tasks per node. This is due to the reduction in the amount of communication needed in the second case, as this amount depends on the granularity of the test case.

Table 3. Case study 1: speed-up values for task (c) with OpenMP–MPI on MARCONI Cluster.

OpenMP Threads per Node	10 Nodes 2 MPI per Node	20 Nodes 2 MPI per Node	40 Nodes 1 MPI per Node
1 (reference)	1 (reference)	1.99	1.99
5	4.96	9.85	9.88
10	9.86	19.7	19.7
15	14.8	29.5	23.2

4.3. Case Study 2: Fusion Reactor Eddy Currents Analysis

Case study 2 is taken from a typical eddy current problem in nuclear fusion machines, associated with the fast electromagnetic transients produced by disruption of the confined plasma. The reference machine is the fusion reactor ITER [18]. By exploiting the symmetry and periodicity of the problem, the computational domain can be limited to a sector of 40° of the tokamak (Figure 5a). The main conducting components falling into such a sector have been modelled, such as the vacuum vessel, the thermal shield, the coils, and the supports, etc. The 3D mesh consists of 80,573 elements, 138,762 nodes, and N_{dof} = 106,544 DoFs. As pointed out, the boundary conditions impose cyclic symmetry at ±20°.

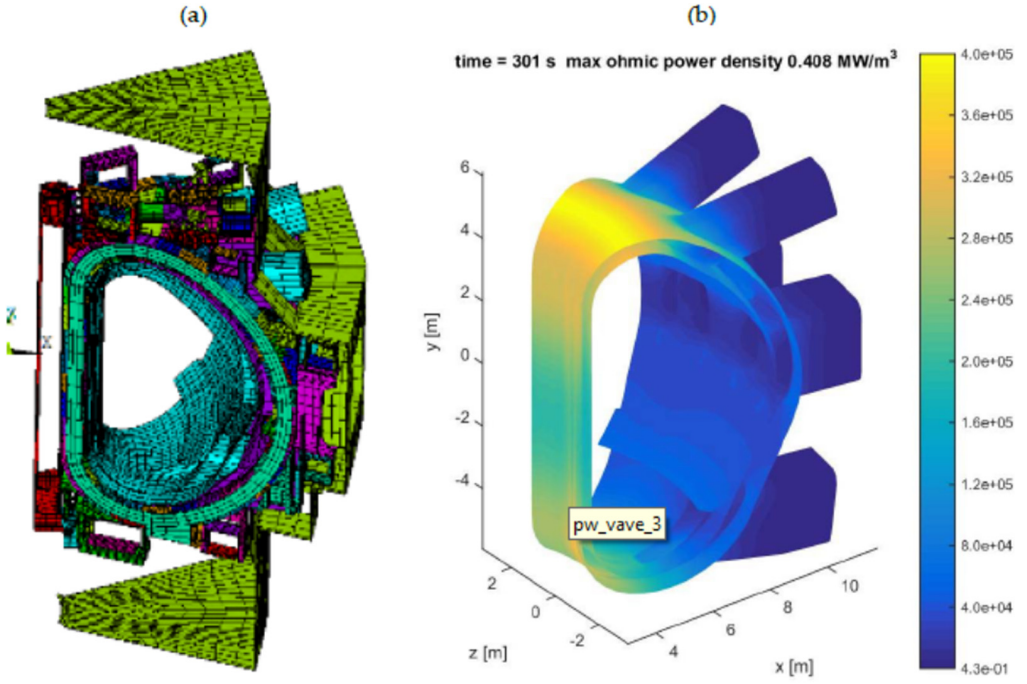

Figure 5. Case study 2: (**a**) a 40° sector of ITER magnet system; (**b**) distribution of ohmic power density at a given time instant during the considered transient event.

Table 4 provides the computed costs of the assembly of $\boldsymbol{L}$ (task b). As a reference case, here we have assumed the choice of 22 OpenMP threads and 1 MPI task per node. The reference time is 16,064 s. Table 5 reports the speed up values for the inversion of $\boldsymbol{Z}$ (task d), with the reference solution given by 22 OpenMP threads and 2 MPI tasks per node, with 2 nodes. The reference time is here 907 s. Once again, the behavior of the speed-up scale is quite linear when increasing the number of nodes, with a limited degradation. Comparing the reference time for task b (assembly of **L**) and task d (inversion), we observe that this case study is another example where the inversion time is much smaller than the assembly one, as already pointed out in Section 2 and in discussing the benchmark case study. In addition, the numerical solution of problems such as those in case studies 1 and 2 exploits the symmetry of the machines, thus limiting the solution domain to a sector. Moreover, the periodic symmetric conditions require considering n replicas of the DoF

during the assembly phase, thus increasing the corresponding time, without affecting the inversion time.

Table 4. Case study 2: speed-up values for task b, with OpenMP–MPI on MARCONI Cluster.

OpenMP Threads per Node	Nodes 1 MPI per Node	Speed-Up
22 (reference)	1 (reference)	1.0
22	25	23.9
22	36	33.1
22	64	53.5

Table 5. Case study 2: speed-up values for task d with OpenMP–MPI on MARCONI Cluster.

OpenMP Threads per Node	Nodes 2 MPI per Node	Speed-Up
22 (reference)	2 (reference)	1.0
11	8	2.4
22	8	4.3
11	18	4.9
22	18	7.9

5. Conclusions

A hybrid parallel computation approach implementing the joint use of MPI and OpenMP paradigms is here implemented and applied to challenging numerical magneto-quasi-static (MQS) problems. The approach has been used to parallelize the assembly of the inductance matrix: it has been shown that this task may be cost demanding as well as involving the task of matrix inversion, when high accuracy is required in the solution of MQS problems (for instance, when using 2 or more integration Gauss points). The pure MPI and the hybrid OpenMP–MPI approaches have been theoretically compared in terms of their computational performance in assembling such a matrix. The hybrid OpenMP–MPI approach saves resources (since it requires a lower number of nodes for a fixed computational time) and provides better performance (it requires lower times for fixed number of nodes). The analyzed case studies demonstrate speed-up values up to about $50\times$.

The proposed approach has a potential great impact on the efficient use of parallel computational resources, suggesting an efficient handling of tasks and threads, to avoid resources (cores and/or ram) being idle while the other processes are running elsewhere.

Author Contributions: Conceptualization, S.V.; methodology, A.C., A.M., G.G., F.C. and S.V.; software and validation, F.C. and S.V.; writing, A.C. and A.M.; supervision, F.V. All authors have read and agreed to the published version of the manuscript.

Funding: The work was partially supported by the Projects: "Smart Distributed Systems" under the program "Dipartimenti di Eccellenza 2018–2022", and the program PRIN, grant # 20177BZMAH, funded by MIUR, Italian Ministry of University and Research, and by the project "LargEM", 3rd Marconi Fusion cycle.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rubinacci, G.; Tamburrino, A.; Ventre, S.; Villone, F. A Fast Algorithm for Solving 3D Eddy Current Problems with Integral Formulations. *IEEE Trans. Magn.* **2001**, *37*, 3099–3103. [CrossRef]
2. Rubinacci, G.; Tamburrino, A.; Ventre, S.; Villone, F. A fast 3-D multipole method for eddy-current computation. *IEEE Trans. Magn.* **2004**, *40*, 1290–1293. [CrossRef]
3. Hackbusch, W. A sparse matrix arithmetic based on H-matrices. Part I: Introduction to H-matrices. *Computing* **1999**, *62*, 89–108. [CrossRef]
4. Ma, T.; Bosilca, G.; Bouteiller, A.; Dongarra, J.J. Kernel-assisted and topology-aware MPI collective communications on multicore/many-core platforms. *J. Parallel Distrib. Comput.* **2013**, *73*, 1000–1010. [CrossRef]

5. Klinkenberg, J.; Samfass, P.; Bader, M.; Terboven, C.; Müller, M.S. CHAMELEON: Reactive Load Balancing for Hybrid MPI+OpenMP Task-Parallel Applications. *J. Parallel Distrib. Comput.* **2020**, *138*, 55–64. [CrossRef]
6. Legrand, A.; Renard, H.; Robert, Y.; Vivien, F. Mapping and load-balancing iterative computations. *IEEE Trans. Parallel Distrib. Syst.* **2004**, *15*, 546–558. [CrossRef]
7. The MPI Forum. MPI: A Message Passing Interface. In Proceedings of the Supercomputing '93: 1993 ACM/IEEE Conference on Supercomputing, Portland, OR, USA, 15–19 November 1993; pp. 878–883.
8. Dagum, L.; Menon, R. Openmp: An industry-standard API for shared memory programming. *Comput. Sci. Eng.* **1998**, *1*, 46–55. [CrossRef]
9. The OpenMP® API Specification for Parallel Programming. Available online: https://openmp.org/wp/about-openmp (accessed on 20 September 2021).
10. Saczek, M.; Wawrzak, K.; Tyliszczak, A.; Boguslawski, A. Hybrid MPI/Open-MP acceleration approach for high-order schemes for CFD. *J. Phys. Conf. Ser.* **2018**, *1101*, 012031. [CrossRef]
11. Ahn, J.M.; Kim, H.; Cho, J.G.; Kang, T.; Kim, Y.-S.; Kim, J. Parallelization of a 3-Dimensional Hydrodynamics Model Using a Hybrid Method with MPI and OpenMP. *Processes* **2021**, *9*, 1548. [CrossRef]
12. Procacci, P. Hybrid MPI/OpenMP Implementation of the ORAC Molecular Dynamics Program for Generalized Ensemble and Fast Switching Alchemical Simulations. *J. Chem. Inf. Model.* **2016**, *56*, 1117–1121. [CrossRef] [PubMed]
13. Sataric, B.; Slavnić, V.; Belic, A.; Balaz, A.; Muruganandam, P.; Adhikari, S. Hybrid OpenMP/MPI programs for solving the time-dependent Gross-Pitaevskii equation in a fully anisotropic trap. *Comput. Phys. Commun.* **2016**, *200*, 411. [CrossRef]
14. Jiao, Y.Y.; Zhao, Q.; Wang, L.; Huang, G.-H.; Tan, F. A hybrid MPI/OpenMP parallel computing model for spherical discontinuous deformation analysis. *Comput. Geotech.* **2019**, *106*, 217–227. [CrossRef]
15. Migallón, H.; Piñol, P.; López-Granado, O.; Galiano, V.; Malumbres, M.P. Frame-Based and Subpicture-Based Parallelization Approaches of the HEVC Video Encoder. *Appl. Sci.* **2018**, *8*, 854. [CrossRef]
16. Xu, Y.; Zhang, T. A hybrid open MP/MPI parallel computing model design on the SMP cluster. In Proceedings of the 6th International Conference on Power Electronics Systems and Applications, Hong Kong, China, 15–17 December 2015.
17. Shen, Y.; Cao, C. Parallel method of parabolic equation for electromagnetic environment simulation. In Proceedings of the IEEE Information Technology, Networking, Electronic and Automation Control Conference, Chongqing, China, 20–22 May 2016; pp. 515–519.
18. Guo, H.; Hu, J.; Nie, Z.P. An MPI-OpenMP Hybrid Parallel H -LU Direct Solver for Electromagnetic Integral Equations. *Intern. J. Antennas Propag.* **2015**, *2015*, 615743. [CrossRef]
19. Wuatelet, P.; Lavallee, P.-F. *Hybrid MPI/OpenMP Programming, PATC/PRACE Course Material, IDRIS/MdlS*; The Partnership for Advanced Computing in Europe: Barcelona, Spain, 2015.
20. Barney, B. *Introduction to Parallel Computing*; Livermore National Laboratory: Livermore, CA, USA, 2015.
21. Albanese, R.; Rubinacci, G. Finite Element Methods for the Solution of 3D Eddy Current Problems. *Adv. Imaging Electron. Phys.* **1998**, *102*, 1–86.
22. Wu, X.; Taylor, V. Performance Modeling of Hybrid MPI/OpenMP Scientific Applications on Large-scale Multicore Supercomputers. *J. Comput. Syst. Sci.* **2013**, *79*, 1256–1268. [CrossRef]
23. EFDA, European Fusion Development. The ITER Project. Available online: www.iter.org (accessed on 15 September 2021).
24. Scalable Linear Algebra PACKage. Available online: www.scalapack.org (accessed on 12 September 2021).
25. Albanese, R.; Rubinacci, G. Integral formulation for 3D eddy-current computation using edge elements. *IEE Proc. A* **1988**, *135*, 457–462. [CrossRef]
26. Rubinacci, G.; Fresa, R.; Ventre, S. An Eddy Current Integral Formulation on Parallel Computer Systems. *Intern. J. Numer. Methods Eng.* **2005**, *62*, 1127–1147.
27. Marathe, J.; Nagarajan, A.; Mueller, F. Detailed cache coherence characterization for openmp benchmarks. In Proceedings of the 18th Annual International Conference on Supercomputing, Malo, France, 26 June–1 July 2004.
28. MARCONI, the Tier-0 System. Available online: www.hpc.cineca.it/hardware/marconi (accessed on 15 September 2021).