Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Evaluation of Multi-threaded OpenFOAM Hybridization for Massively Parallel Architectures

Paride Dagna[a]*, Joerg Hertzer[b]

[a] *CINECA-SCAI Department, Via R. Sanzio 4, Segrate (MI) 20090, Italy*
[b] *HLRS, Nobelstr. 19, D-70569 Stuttgart, Germany*

Abstract

The performance results from the hybridization of the OpenFOAM linear system solver, tested on the CINECA Fermi and the HLRS Hermit supercomputers are presented in this paper. A comparison between the original and the hybrid OpenFOAM versions on four physical problems, based on four different solvers, will be shown and a detailed analysis of the behavior of the main computing and communication phases, which are responsible for scalability during the linear system solution, will be given.

1. Introduction

The OpenFOAM® (Open Field Operation and Manipulation) CFD Toolbox is a free, open source CFD software package produced by OpenCFD Ltd. It has a large user base across most areas of engineering and science, from both commercial and academic organizations. OpenFOAM has an extensive range of features to solve anything from complex fluid flows involving chemical reactions, turbulence and heat transfer, to fluid dynamics and electromagnetics.

By being open source, OpenFOAM offers users complete freedom to customize and extend its existing functionality. From a general point of view, it can be thought as a framework where Computational Fluid Dynamics (CFD) programmers can build their own code, as it provides them with the abstraction sufficient to think of a problem in terms of the underlying mathematical model [1].

This key feature fostered the wide acceptance of the code into the CFD community within both academic and industrial groups.

As outlined in several papers, OpenFOAM is a MPI-parallelized application whose performance, scalability and

* Corresponding author. *E-mail address*: p.dagna@cineca.it

parallel efficiency have been tested across many different architectures, including massively parallel clusters.

To increase scalability, the parallelization strategy employs the "Zero-Halo Layer Decomposition" approach whereby a given volume of cells is assigned to each MPI process and communication is only with neighbouring cells. As the number of subdomains increases and consequently the number of cells within each domain decreases, it will clearly result in a higher number of neighbouring cells, which requires more communication between processes, needed to exchange information in the halo regions. As shown in literature, according to the benchmarks on some of the most commonly used OpenFOAM solvers, the MPI communications consistently dominate the simulation time going over some hundreds of cores.

This paper stems from the the work started by Massimiliano Culpo [2], where a hybrid multi-threaded solution was introduced, with the intention of reducing MPI communications and increasing scalability. In this paper, the hybrid multi-threaded solution has been extended to four OpenFOAM solvers, and its behaviour on the Fermi [3] and Hermit [4] systems, with the relative test cases, results and performance considerations, will be shown. The numbers presented in section 4 refer to the BG/Q system: during the testing phase it was found that the two machines presented almost the same behaviour for the purpose of this study, where we intend to show if solver hybridization could improve scalability.

2. Tier-0 system specifications and installation notes

All the tests that will be presented in this document have been executed on Fermi, a Tier-0 machine which is at present CINECA's main HPC facility; the first and the last test have been executed on Hermit too, which is the main HPC system hosted at HLRS.

Fermi is an IBM BlueGene/Q system composed of 10.240 PowerA2 sockets running at 1.6GHz, with 16 cores each, totalling 163840 compute cores and a system peak performance of 2.1 PFlop/s. The interconnection network is a very fast and efficient 5D Torus. Fermi is one of the most powerful machines in the world, and was ranked #9 in the top 500 supercomputer sites list published in November 2012.

The version of OpenFOAM used is the 2.1.1 build with GNU compilers 4.4.6 and BG/Q system proprietary MPI. The code has been profiled using TAU Performance System® analyzer [5].

Hermit is a Cray XE6 supercomputer composed of 6276 AMD Opteron Interlagos sockets running at 2.3GHz, with 16 cores each, totalling 113664 compute cores and a system peak performance of about 1 PFlop/s. The interconnection is the high speed network CRAY Gemini.

The version of OpenFOAM used is the 2.1.1 build with the CRAY C++ compiler and a system proprietary MPI. The code has been profiled using the Integrated Performance Monitoring analyser [6].

3. Hybridization strategy

Following the guidelines presented in [2], hybridization using OpenMP directives in linear algebra libraries, PCG solver and DIC, diagonal and DILU preconditioners, has been implemented on OpenFOAM-2.1.1.

The main computing phases executed by the PCG module for the solution of the linear system, including preconditioning and linear algebra operations, are represented in the flow chart of figure 1. The right hand side of the diagram shows an outline of the hybridization strategy for the linear system solution.

These operations that follow the Zero-Halo Layer approach are defined in the `solve` method in the `PCG.C` source code, and can be categorised in five main phases which are repeated until convergence criteria are reached:

- preconditioning
- scalar product and reduction of the partial sums (requires MPI communication)
- cells update
- matrix–vector multiplication and reduction of the partial results (requires MPI communication)

- solution and residual update (requires MPI communication)

The only hybridizable parts, where OpenMP directives were introduced, are preconditioning, matrix–vector multiplication and cells update. For the other operations, where data exchange between processors is required, hybridization is not feasible and execution is handed over to the master thread while the others wait for the master at the OpenMP barrier.

Only the diagonal preconditioner was fully hybridized, while the others were only partially hybridized because some internal parts created data dependence issues. Also the two cycles for cells update were fully hybridized.

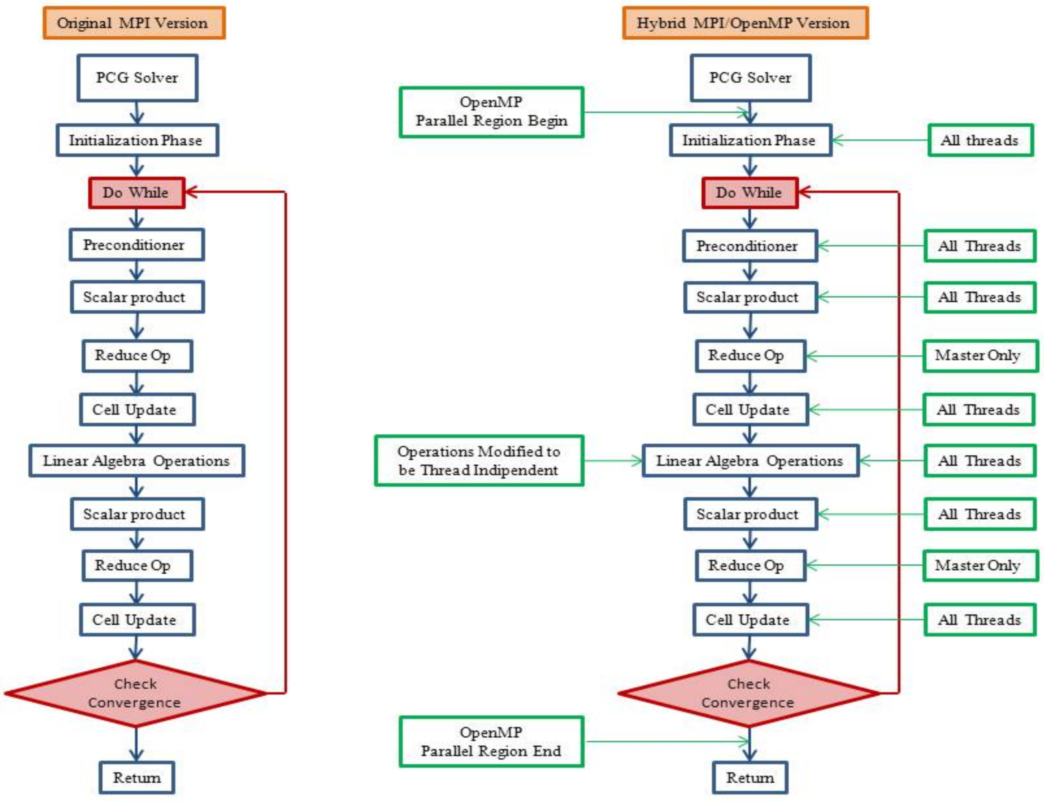

Fig. 1. Flow chart of the main phases for linear system solution. The diagram on the right is an outline of the hybridization strategy

4. Test cases and results

In this section the selected test cases and the related performance and scalability results obtained with the original pure MPI and the hybrid version of OpenFOAM will be presented.

The test cases have been selected in order to show the performance of the four hybridized solvers (*icoFoam, simpleFoam, interFoam, coldengineFoam*) respect to the original pure MPI ones.

Moreover, the complexity of the mesh, relative to each benchmark, and the different solvers involved, change the way communication is done and the overall number of MPI calls.

In order to more accurately assess the performance of the two versions during the solution phase, all simulations were carried out by turning off the saving of solution feature during intermediate time steps to avoid possible noise due to I/O operations. To evaluate the performance of the computing phase, the time to execute the first iteration, which includes also the initialization processes, is not considered, and all the timings listed in the tables in this section represent the time taken between the last and the end of the first iteration.

4.1. Lid-driven cavity flow

The lid-driven cavity flow benchmark [7] involves the solution of a laminar flow over a three-dimensional structured uniform cubic domain using the *icoFoam* solver. The flow is assumed to be isothermal and incompressible. All the boundaries of the cube are static walls, with the exception of the top one that moves in the *x* direction.

The dimension of the mesh is of 200x200x200 cells. For the subdivision of the domain for a given number of processors, the simple [8] algorithm was chosen.

The times listed in the table below refer to 300 time steps.

Table 1. Lid-driven cavity flow. Comparison between the original pure MPI and the hybrid version. Timings listed in the tables in this section represent the time taken between the last and the end of the first iteration

Cores #	Original Pure MPI (sec)	Hybrid Configuration		Hybrid MPI OpenMP (sec)	Pure MPI vs Hybrid %	Faster Solution	
		MPI tasks	OpenMP threads			Pure MPI	Hybrid MPI + OpenMP
64	18772	64	0	27360	31	√	
128	9380	64	2	15660	40	√	
256	4770	64	4	10080	53	√	
512	2706	64	8	7230	63	√	
1024	1920	1024	0	2040	6	√	
2048	990	1024	2	1320	25	√	
4096	720	1024	4	960	25	√	
4096	720	2048	2	840	14	√	

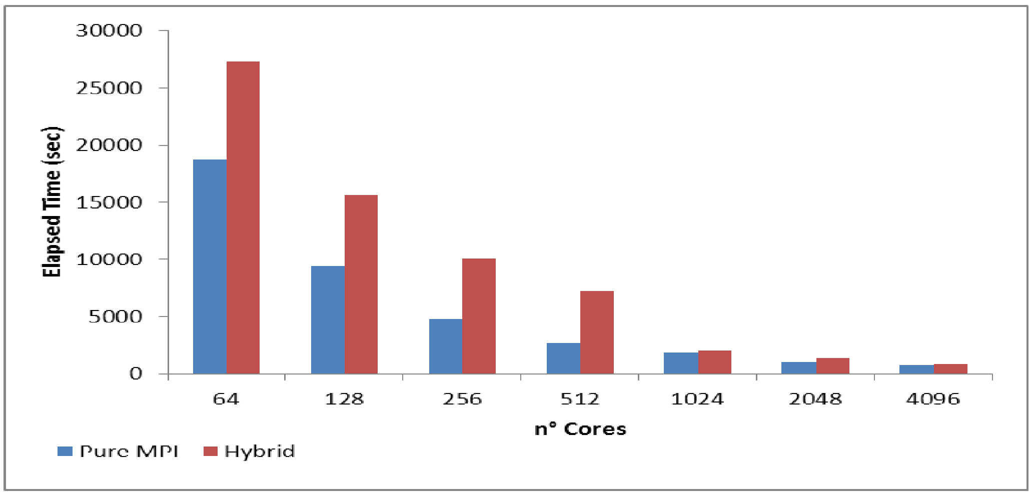

Fig. 2. Elapsed time of pure MPI and mixed OpenMP + MPI solutions.

Referring to table 1, in figure 2 is shown the simulation time, taken between the last and the end of the first iteration, obtained from pure MPI runs and the best mixing of MPI tasks and OpenMP threads. Looking at the benchmark's results it comes out that the original pure MPI version of OpenFOAM is faster in all the configurations tested, even when the number of MPI processes and consequently of MPI communications becomes considerably high. In chapter 5 the reasons of the poorer performance of the hybrid version during the linear system solution will be investigated in detail.

The tests at HLRS also showed no real improvement by hybridization, although initial runs with two OpenMP threads in each MPI process indicated some potential improvement. However, additional tests showed that the same effect was gained by runs without OpenMP, when every second core was left unused and the remaining ones were used by pure MPI processes. This might be explained by memory bandwidth bottlenecks. Because this effect is not caused by multi-threaded hybridization, further investigation was not a subject of this paper.

4.2. NACA Airfoil

The NACA [9] airfoil benchmark involves the solution of a turbulent flow over a three-dimensional structured hexahedral mesh using the *simpleFoam* solver. The flow is assumed to be isothermal and incompressible.

The dimension of the mesh is of almost 9000000 cells. For the subdivision of the domain for a given number of processors, the simple [8] algorithm was chosen. The times listed in table 2 below refer to 100 time steps.

Table 2. NACA Airfoil. Comparison between the original pure MPI and the hybrid version. Timings listed in the tables in this section represent the time taken between the last and the end of the first iteration.

Cores #	Original Pure MPI (sec)	Hybrid Configuration		Hybrid MPI OpenMP (sec)	Pure MPI vs Hybrid %	Faster Solution	
		MPI tasks	OpenMP threads			Pure MPI	Hybrid MPI + OpenMP
64	1418	64	0	1726	18	√	
128	903	64	2	1259	28	√	
256	464	64	4	977	53	√	
512	297	512	0	367	19	√	
1024	199	512	2	259	23	√	
2048	138	512	4	209	34	√	

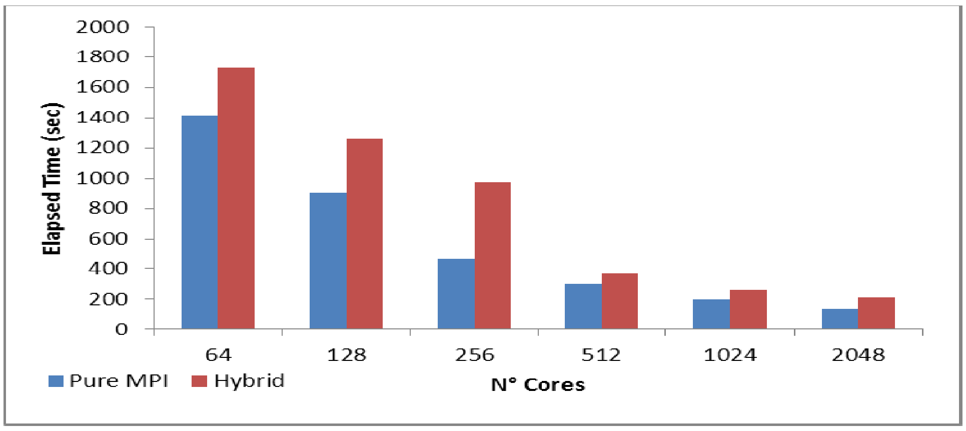

Fig. 3. Elapsed time of pure MPI and mixed OpenMP + MPI solutions.

Referring to table 2, in figure 3 is shown the simulation time, taken between the last and the end of the first iteration, obtained from pure MPI runs and the best mixing of MPI tasks and OpenMP threads.

As in the lid-driven cavity benchmark, the Pure MPI version remains the fastest for all the configurations tested.

4.3. *DTMB Hull*

The DTMB hull [10] benchmark involves the solution of a turbulent two phase immiscible fluid over a three-dimensional fully unstructured hexahedral mesh using the *interFoam* solver. The flow is assumed to be isothermal and compressible.

The dimension of the mesh is of almost 5500000 cells and for the subdivision of the domain for a given number of processors, the simple [8] algorithm was chosen.

The times listed in the table below refer to 100 time steps.

Table 3. DTMB Hull. Comparison between the original pure MPI and the hybrid version. Timings listed in the tables in this section represent the time taken between the last and the end of the first iteration.

Cores #	Original Pure MPI (sec)	Hybrid Configuration		Hybrid MPI OpenMP (sec)	Pure MPI vs Hybrid %	Faster Solution	
		MPI tasks	OpenMP threads			Pure MPI	Hybrid MPI + OpenMP
64	5072	64	0	6516	22	√	
128	2828	64	2	4292	34	√	
256	1733	64	4	2820	39	√	
512	1279	512	0	1477	13	√	
1024	1954	512	2	1185	-65		√

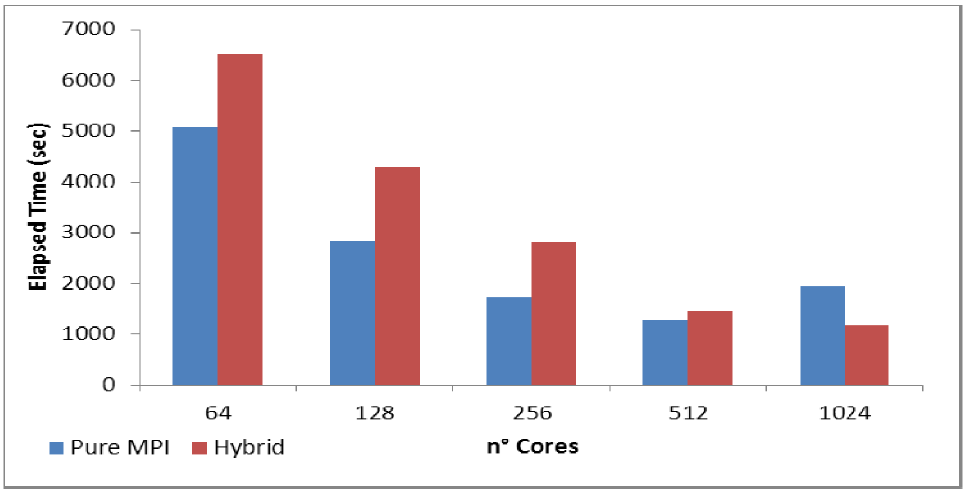

Fig. 4. Elapsed time of pure MPI and mixed OpenMP + MPI solutions.

Referring to table 3, in figure 4 is shown the simulation time, taken between the last and the end of the first iteration, obtained from pure MPI runs and the best mixing of MPI tasks and OpenMP threads.

Looking at the results in the table above we can see the in the last row the hybrid version is faster but the gain obtained with respect to the pure MPI version with 512 cores is only 7%, which is not an acceptable value considering that we have doubled the number of cores. The reasons why the hybrid version in the configuration of the last row in table 3 is faster than the pure MPI one will be explained in chapter 5.

4.4. Cold flow

The benchmark is an industrial test case which involves cold flow LES simulation of a turbulent flow over a three-dimensional block structured mesh in an engine-like geometry, using the *coldengineFoam* compressible solver [11].

The dimension of the mesh is of almost 4500000 cells and for the subdivision of the domain for a given number of processors, the simple [8] algorithm was chosen.

Table 4. Cold flow. Comparison between the original pure MPI and the hybrid version. Timings listed in the tables in this section represent the time taken between the last and the end of the first iteration.

Cores #	Original Pure MPI (sec)	Speed up	Hybrid Configuration		Hybrid MPI OpenMP (sec)	Pure MPI vs Hybrid %		
			MPI tasks	OpenMP threads			Pure MPI	Hybrid MPI + OpenMP
64	36209	64	64	0	47790	24	√	
128	14421	161	64	2	39010	63	√	
256	7966	291	64	4	20464	61	√	
512	4806	482	512	0	6102	21	√	
1024	4268	543	512	2	4380	3	√	

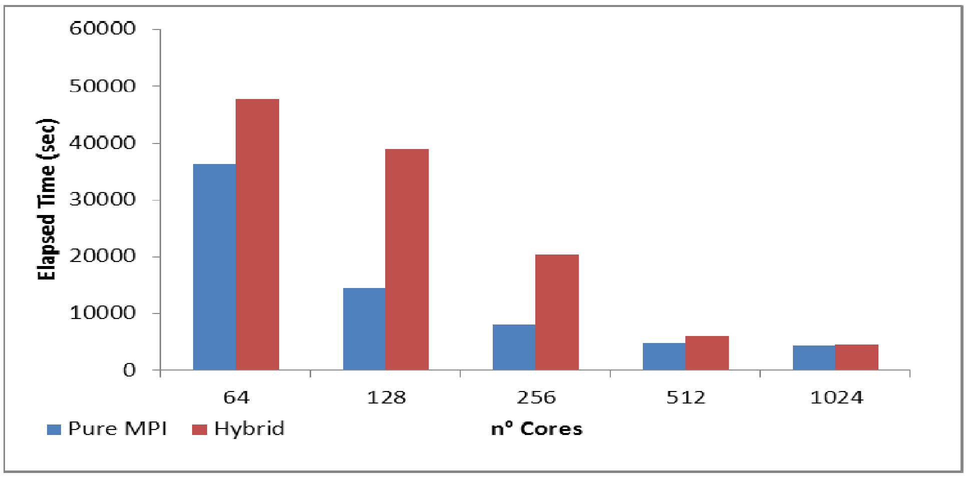

Fig. 5. Elapsed time of pure MPI and mixed OpenMP + MPI solutions.

Referring to table 4, in figure 5 is shown the simulation time, taken between the last and the end of the first iteration, obtained from pure MPI runs and the best mixing of MPI tasks and OpenMP threads.

In table 4 the “Speed up” column for the pure MPI version has been inserted because in this test a super-linear speed-up effect seems to appear. Indeed this effect is due to the fact that the total number of iterations needed for the solution to reach convergence during the whole simulation, is about 1700 for the case with 64 cores and only about 1050 for the configurations with 128, 256 and 512 cores. In the test with 1024 cores we can see a drop in performance which is essentially due to higher number of iterations (1200) needed to have solution convergence and the increase of communications with respect to computation time as the number of cells per core is only about 4400.

Similar as described above for the lid-driven cavity flow, the tests at HLRS showed some improvement, when every second core was left unused, but again no additional improvement with OpenMP.

5. Analysis and discussion

Looking at the results listed in section 4, where the original pure MPI version is compared with the hybrid one, the hybrid version shows a poorer performance even when OpenMP is turned off (OpenMP # 0).

This behaviour is essentially due to the reordering of the operations and instructions used in the linear algebra routines to make them thread independent. The number of total operations remains almost the same but the different structure influences negatively the cache usages causing a drop in performance. This behaviour is processor and compiler dependent, in fact, the same test executed on the AMD processors on the HLRS Hermit system at Stuttgart, shows only a 10% or less of performance loss.

In the following charts (fig. 6) the total amount of time spent in MPI calls with respect to the time spent in computing and I/O operations of the four tests above is represented, for the whole simulation time. These data have been obtained with the TAU profiler wrapping the MPI calls using library preloading. In this configuration the overhead induced by the profiler is negligible allowing a for greater accuracy in the results.

For each test, the values in the charts refer to the case with the higher number of cores, which is the more significant one, because it's the one where MPI calls have the major weight and a better performance of the hybrid version should be expected. In fact for the pure MPI version if, for example, we use 1024 cores we will have 1024 MPI tasks running. Using the hybrid version we can run with 1024 cores using, for example 512 MPI task and 2 OpenMP threads, reducing the number of MPI tasks and consequently the amount of MPI communication.

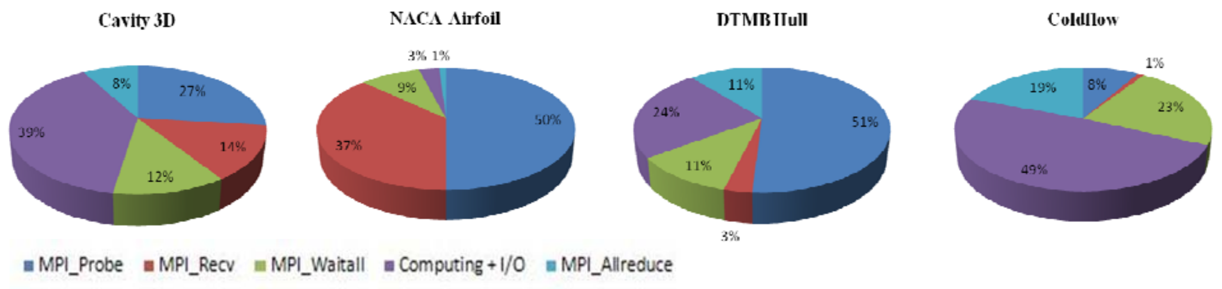

Fig. 6. Percentage of time spent in MPI calls with respect to computing and I/O operations for the whole simulation.

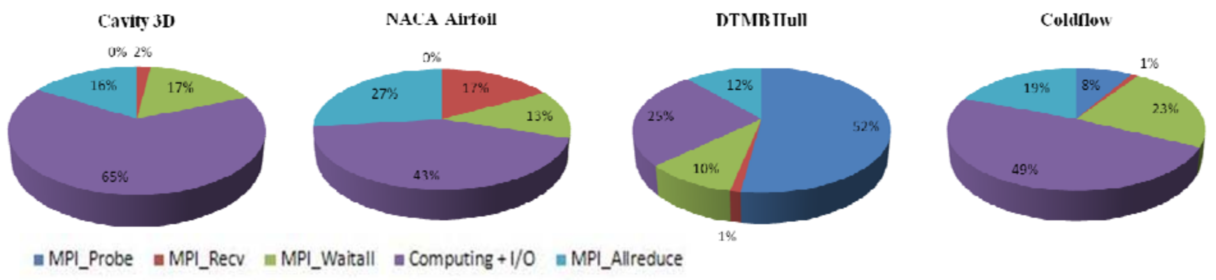

Fig. 7. Percentage of time spent in MPI calls respect to computing and I/O operations for just the computing phase.

Looking at the percentages, MPI seems to dominate over computing, but the information we receive from the graphs of fig 6. is not completely correct because we should focus only on the portion of MPI calls in the computing phase.

The charts reported in fig 7. show the real amount of communication that is present in the solution phase. It's possible to notice immediately some relevant differences from the graphs of fig 6. for the "Cavity 3D" and the "NACA Airfoil" where for both tests the call to *MPI_Probe* disappear, highlighting that this function is only used in the start-up phase, and also the time spent in *MPI_Recv* is significantly reduced.

For the "DTMB Hull" and the "Coldflow" the ratio between communication and computing remains almost the same. These differences among the charts show that the MPI structure changes according to the solver used, making it impossible to find a general schema of the message passing rules among processes.

Also for the cases with the higher number of cores reported in the tables in section 4, the pure MPI version performs better. The only exception is for the "DTMB Hull" at 1024 cores where, adding the percentages relative to MPI calls in fig 7 it comes out that the amount of time spent in message passing is about 75%.

To investigate further the reasons for this behaviour, some timers were inserted in the hot sections of the PCG solver for the pure MPI and hybrid version during the solution of the linear system.

As explained in section 3, the main phases are:

- Preconditioning.
- scalar product and reduction of the partial sums.
- cells update.
- matrix – vector multiplication and reduction of the partial results.
- solution and residual update.

In the table below the time spent per iteration in the phases listed above is shown. Only the results relative to the lid-driven cavity and DTMB Hull tests are shown as similar considerations can be retrieved for the others.

Table 5. Lid-driven cavity. Time per iteration in the main hybridized and not-hybridized parts. Time is in milliseconds

Cores #	Original pure MPI (msec)				Hybrid (msec)					
	Precond	M-V mult	Interface Update	Not Hybridizable	MPI tasks	OpenMP threads	Precond	M-V mult	Interface Update	Not Hybridizable
64	2,50	19,00	3,40	6,30	64	0	2,50	32,00	3,80	6,30
128	1,00	9,80	1,70	3,00	64	2	1,80	15,97	2,10	6,30
256	0,54	5,00	0,90	1,40	64	4	0,54	8,90	9,40	6,30
512	0,27	2,45	0,45	0,70	64	8	0,27	4,70	0,50	6,30
1024	0,15	1,30	0,26	0,39	1024	0	0,17	2,20	0,31	0,39
2048	0,08	0,72	0,14	0,24	1024	2	0,09	1,20	0,19	0,39
4096	0,50	0,44	0,07	0,16	1024	4	0,06	0,77	0,12	0,39
4096	0,50	0,44	0,07	0,16	2048	2	0,03	0,44	0,07	0,24

The numbers above reveal three main constraints to the performance of the hybrid version:

- Greater time spent in linear algebra operations (M-V mult) by the hybrid version because of the issues explained at the beginning of this section.
- The time spent in the non-hybridizable parts, where MPI communications are present, falls as the number of cores increases. This behaviour is essentially related to the decreasing of the subdomains of the initial mesh and the Zero-Halo Layer approach. In the hybrid version, once the number of MPI processes is fixed, the dimension of subdomains, and as a consequence the time spent in non-hybridizable parts, is fixed too, reducing the possible benefits derived from using OpenMP threads.
- For high core numbers, the time per iteration becomes very small, above all for preconditioning and

interface update, making the overhead introduced by the activation of the OpenMP parallel region not negligible.

Table 6. DTMB Hull. Time per iteration in the main hybridized and not-hybridized parts. Time is in milliseconds.

Cores #	Original pure MPI (msec)				Hybrid (msec)					
	Precond	M-V mult	Interface Update	Not Hybridizable	MPI tasks	OpenMP threads	Precond	M-V mult	Interface Update	Not Hybridizable
64	1,41	9,98	3,50	3,10	64	0	1,49	17,28	3,41	3,00
128	0,58	5,11	1,62	1,80	64	2	0,60	9,17	1,70	3,00
256	0,31	2,76	0,83	1,00	64	4	0,31	5,08	0,91	3,00
512	0,16	1,47	0,45	0,42	512	0	0,17	2,32	0,49	0,42
1024	0,08	0,82	0,24	0,24	512	2	0,09	1,15	0,24	0,42

According to the timings in table 3 of section 4, the hybrid version at 1024 cores is faster, for the whole computing phase.

In the table above, where only the timings taken by the Preconditioned Conjugate Gradient solver are reported, even at 1024 cores the pure MPI version remains the faster one. The drop of performance for the original version, when the whole computing phase is accounted, is due to the time spent in the *MPI_Probe* routine as shown from the following TAU profiling images (fig. 8), taken on a long simulation, to increase the reliability of the results. The call to this routine is done outside of the PCG solver.

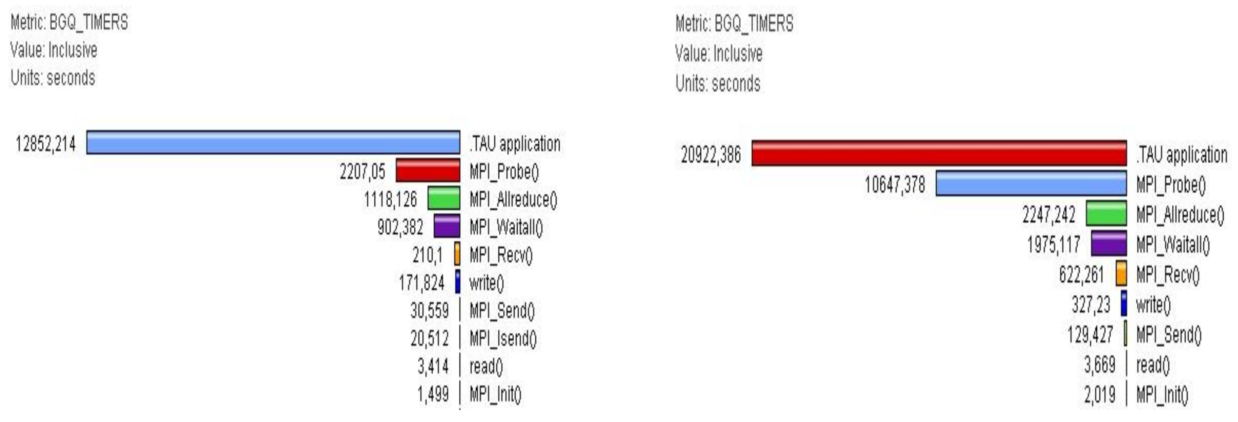

Fig. 8. Total amount of the main MPI calls respect to computing and I/O operations for the dtmb hull. It's evident the increase of the time spent in the MPI_Probe routine going from 512 (on the left) to 1024 cores (on the right).

6. Conclusions

A hybrid OpenFOAM version has been analysed using four solvers and relative benchmarks, characterized with a specific complexity of the mesh, of the problem to be solved and of the amount of MPI communication needed to exchange information among the halo regions of the subdomains.

From the results obtained through the tests in chapter 4 and the behaviour highlighted for the computing and communication phases during the linear system solution (chapter 5), it is clear that obtaining an efficient hybrid version, on supercomputers with a very efficient high speed network, is not so straightforward. Even when communication becomes the limiting factor to scalability, the benefits obtainable from a hybrid implementation are heavily mitigated by the very short execution time per iteration.

As showed in tables 5 and 6 the time spent in “Not hybridisable parts”, where MPI communication is involved, decreases almost proportionally to the increasing of the cores used, reducing the benefits which could come from the hybridization which was really introduced with the idea of giving to OpenMP threads some of the work of MPI tasks in order to reduce domain decomposition and consequently MPI communication.

Anyway as shown in the “DTMB Hull” test case and in fig. 7 for the corresponding test, when MPI communication becomes a real bottleneck to scalability, an hybrid solution can be more performing than a pure MPI one.

For this reason, even if it’s not thinkable to expect a relevant speed-up by hybridization, for complex systems, which may require a lot of communication among MPI processes, hybridization can be a support to scalability.

7. Acknowledgements

This work was financially supported by the PRACE-2IP project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-283493. The work is achieved using the PRACE Research Infrastructure resources Ferni, CINECA, Italy and Hermit, HLRS, Germany. At Hermit software installations required for the tests and the test runs were done by personnel of HLRS.

8. References

1. OpenFOAM main site, http://www.openfoam.com/
2. Massimiliano Culpo, Current Bottlenecks in the Scalability of OpenFOAM on Massively Parallel Clusters, PRACE-1IP Whitepaper http://www.prace-project.eu
3. CINECA, Fermi User Guide. http://www.hpc.cineca.it/content/ibm-fermi-user-guide
4. HLRS, Hermit User Guide. https://wickie.hlrs.de/platforms/index.php/Cray_XE6
5. TAU Performance System® analyzer. http://www.cs.uoregon.edu/research/tau/home.php
6. IPM Integrated Performance Monitoring. http://ipm-hpc.sourceforge.net/
7. Lid-driven cavity flow. http://www.openfoam.org/docs/user/cavity.php
8. The SIMPLE algorithm in OpenFOAM. http://openfoamwiki.net/index.php/The_SIMPLE_algorithm_in_OpenFOAM
9. NACA Airfoil. http://www.pointwise.com/glyph/airfoilGen/
10. DTMB Hull. http://www.hpc.cineca.it/sites/default/files/14_Hydrodynamic_application_for_hull_Penza.pdf
11. F. Piscaglia, A. Montorfano, A. Onorati, Towards the LES simulation of IC Engines with parallel topologically changing meshes, SAE Int. J. Engines 6(2):926-940, 2013