Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Performance Analysis of Fluid-Structure Interactions using OpenFOAM

Michael Moyles[a], Peter Nash, Ivan Girotto

Irish Centre for High End Computing, Grand Canal Quay, Dublin 2

Abstract

The following report outlines work undertaken for PRACE-2IP. The report will outline the computational methods used to examine petascaling of OpenFOAM on the French Tier-0 system CURIE. The case study used has been provided by the National University of Ireland, Galway (NUIG). The profiling techniques utilised to uncover bottlenecks, specifically in communication and file I/O within the code, will provide an insight into the behaviour of OpenFOAM and highlight practices that will be of benefit to the user community.

1. Summary of Model

OpenFOAM is a powerful open source field manipulation tool originally designed for use in Computational Fluid Dynamics (CFD). While its applications have varied in recent years, its main use remains within the field of CFD. Its operation is composed of solvers, dictionaries and domain manipulation tools. It is a popular package amongst academic and industrial users not only because as it allows users edit built-in routines but also due to it's versatility and customisable methods.

OpenFOAM is a MPI-parallelized application which employs a classic domain decomposition strategy, whereby a given volume of cells is assigned to each MPI process It is anticipated there will be an optimal number of cells per core providing optimal walltime. Assuming communication is only with neighbouring cells, a lower number of cells within each domain will clearly result in a higher volume of communication calls between processes... This model is an ideal candidate for benchmarking on PRACE Tier-0 machines as its domain can be cut into numerous segments while still maintaining a significant number of cells per domain.

The model used in this study has been provided by the National University of Ireland, Galway (NUIG). The vessel at its core is the *RV Celtic Explorer* – an Irish based research vessel. The model constructed is solved within a domain containing 421,875 cells (75 in each direction) – a solution space which is a factor of approximately 330 larger than the well-studied *motorbike* tutorial provided by OpenFOAM. The simulations employ the computationally intensive Large Eddy Simulation scheme. Although this scheme is more computationally intensive, it provides a much higher resolution of the turbulent length scale which is of key interest to the research group. The *Celtic Explorer* case study provides an opportunity to analyse a model in current academic use, thus providing an insight into bottlenecks faced by current researchers in the field of CFD. Results obtained here are expected to be highly beneficial to the OpenFOAM user community.

[a] michael.moyles@ichec.ie

The case reported here is described as follows;

- The 3D mesh is rectangular. It is 160m in length (x-dir) and 64m in both width and height (y- & z- dir). It contains 421,875 cells - 75 in each direction. Importantly, this results in a cell size of approx. 2.1m * 0.85m * 0.85m and an aspect ratio of 2.5:1:1. (We will return to this ratio later)
- The size of the vessel is limited to occupy just 1% of the inlet area (yz-plane) – a method employed by the research group to keep the model consistent with previous studies.

The construction and solution of the model is performed using the following sequence of functions

i. *blockMesh* – constructs the 3D domain defined by *blockMeshDict*
ii. *decomposePar* – decomposes the domain into a geometry described by *decomposeParDict* and assigns one domain per process
iii. *snappyHexMesh* – 'snaps' the blockMesh to the *CelticExplorer.stl* input
iv. *reconstructParMesh* – reconstructs the mesh
v. *decomposePar* – decomposes the domain as previous
vi. *interFoam* – a solver for two isothermal, incompressible, immiscible fluids. This solves the Navier-Stokes equations within each domain
vii. *reconstructPar* – reconstruct the entire model

When run sequentially, the majority of the execution time is dominated by the solver, interFoam as expected but a significant amount of time is also consumed by snappyHexMesh. These are therefore the tools that are run in parallel in our studies.

2. Preliminary Scaling Tests

This section outlines a series of scalability tests performed on the case study. While some tests were performed on the *snappyHexMesh* utility our attention was focused on the solver, *interFoam*. These tests involved altering the domain decomposition in *decomposeParDict* and executing the two binaries over the relevant number of cores. We used the profiling tools IPM and Darshan throughout the course of our investigations – descriptions of how these can be linked to OpenFOAM can be found in the appendices. IPM is a portable profiling infrastructure (2) that is particularly useful in building a communication profile of a parallel application. Darshan is a lightweight I/O profiler (1).

2.1 Preliminary Findings

In the following section we will present results from the various tests performed using the profilers outlined above. IPM will provide us with a communication profile of the code whilst Darshan will provide an I/O profile, which as we will see provides us with a more relevant set of results and possible pathway to improve scalability.

2.1.1 IPM

Figure 1 shows data from the IPM profiler. As can be seen the scalability of the *interFoam* solver is poor for the given input settings[b]. Furthermore the communication is disproportionately high for an application to run efficiently on a petascale machine. More insight can be gained into the MPI calls within OpenFOAM by looking further into the IPM output (***Figure 2***, ***Figure 3***). It is clearly seen that *MPI_Recv, MPI_Waitall and MPI_Allreduce* consistently dominate the time spent in communication.

[b] the *interFoam* solver writes to disk 40 times during the course of its execution

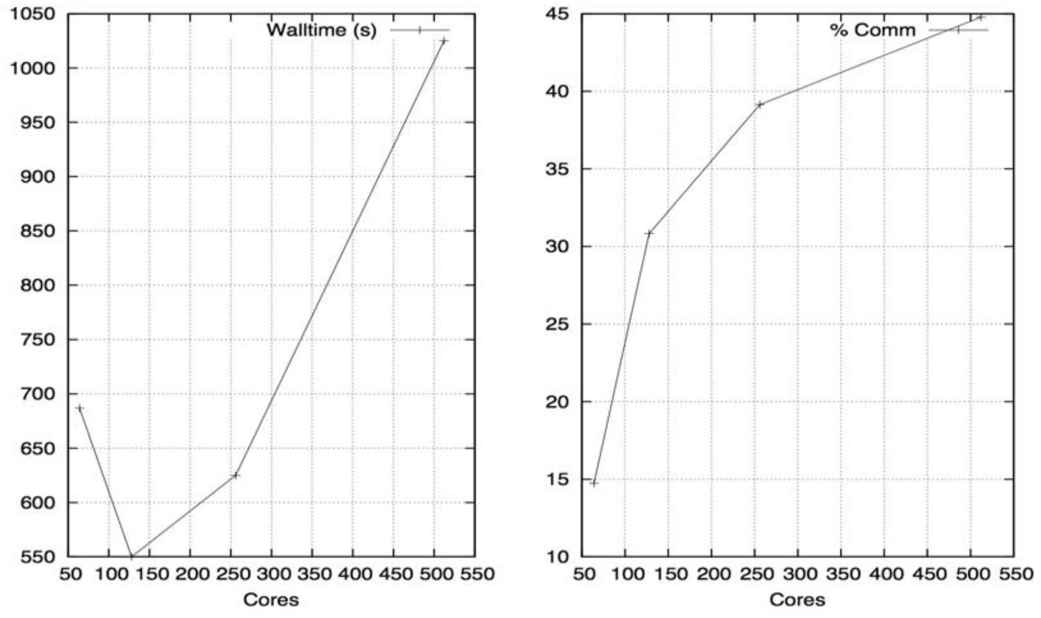


Figure 1: Scaling and Communication Profile for interFoam solver obtained from Curie up to 512 cores

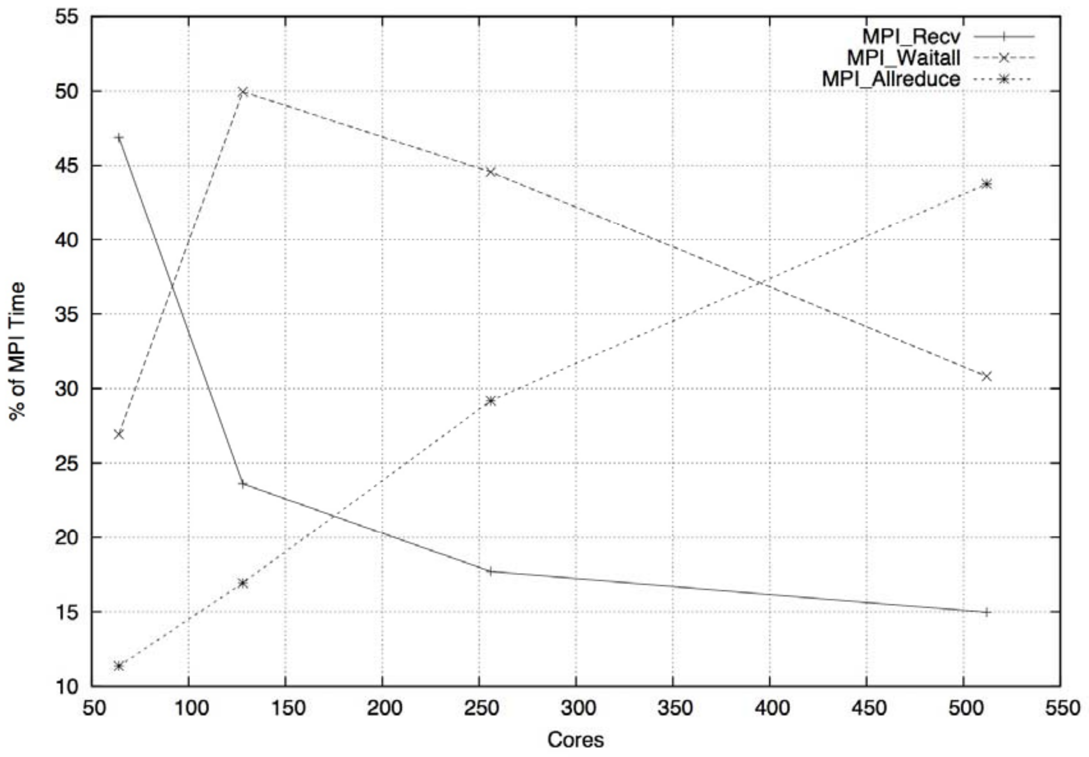


Figure 2: Dominant MPI functions for interFoam solver

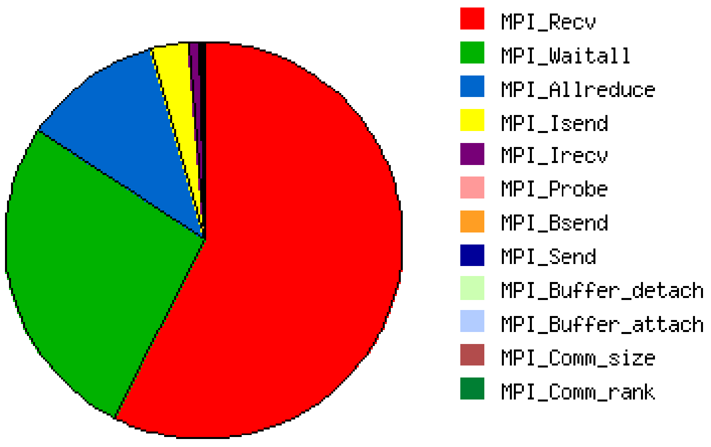


Figure 3: Sample Breakdown of MPI calls for interFoam over 64 cores on Curie

2.1.2 Darshan

The data collected by the Darshan instrumentation of OpenFOAM provides us with a profile of the I/O behaviour. The results are shown in ***Table 1***. We can see that the compute time increases with increasing number of processes with almost a doubling for the last test case with 1024 processes. Metadata handling (reading and writing of files not related to final output) not only takes a large share of overall compute time, but the time used on metadata handling grows disproportionately compared to the time used on compute. We can also see in ***Table 1*** that while the size of the files decreases, the number of files created and read by the solver increase as the number of cores increase. The scalability of this test was even poorer again as we increased the time of the simulation. This is shown in ***Figure 7***. Nevertheless, the improved scaling behaviour breaks down after 256 cores and prior to that the scaling is less than what would be deemed “good” scaling behaviour.

The work of Lindi (4) has found, through inspection of the OpenFOAM source code, that processes communicate with one another through reading and writing of files. Through an instance of an object called FileMonitor, the function *stat()* checks timestaps of metadata files. As can be seen the number of calls to *stat()* increases with the number of cores. This, in turn, increases the time spent handling metadata and therefore the walltime required by *interFoam*.

#Cores	Walltime (s)	Metadata Handling (s)	% of I/O in Simulation	# of Files Created	# of Files Read	Average File Size (K)	# of Stat() calls
64	686	64	9.3	512	1089	597	5E05
128	801	202	25	1024	2177	317	1E06
256	890	274	31	2048	4353	163	2E06
512	1161	389	34	4096	9729	84	4.4E06
1024	2248	892	39	8192	17409	47	8.5E06

Table 1: Darshan Profile[c] for interFoam solver on Curie

[c] The slight discrepancy in walltime between this table and the results shown in ***Figure 1*** is due to the implementation of different profilers (IPM & Darshan)

2.2 Preliminary Conclusions

Early profiling of this case study in OpenFOAM has shown a very poor communication structure. Processes communicate through read and write calls to disk (4) a known scaling pitfall as one process can easily finds itself waiting while the relevant files are written by another process. As can be seen from ***Table 1***, the time spent in reading and writing of these files is significant. Additionally, it has been shown from the IPM profiling, ***Figure 1*** and ***Figure 2*** that an even larger portion of the walltime is spent in MPI calls. The presence of the *MPI_Allreduce* function, ***Figure 2***, can be seen to consume a higher proportion of the communication time (consequently walltime) for larger numbers of cores. This is not surprising behaviour as the function must gather information from all processes, operate on it and then distribute a result back to all processes, an operation which will be more costly when operated over larger numbers of processes.

Figure 4 compares I/O and communication as portions of the execution walltime. They can be seen to follow the similar profiles and their combined effect clearly kills any hope of good scaling behaviour. In the region of 512 cores or more, the remaining portion of time left for the actual solver calculations is seen to be a maximum of 26% of the overall execution time. An extension of this calculation shows that the remaining solver computing time actually drops from approximately 521s to 301s (from ***Table 1***).

These results describe an application which contains the potential to scale well but poor I/O design along with a heavy communication strategy impede any performance gains.

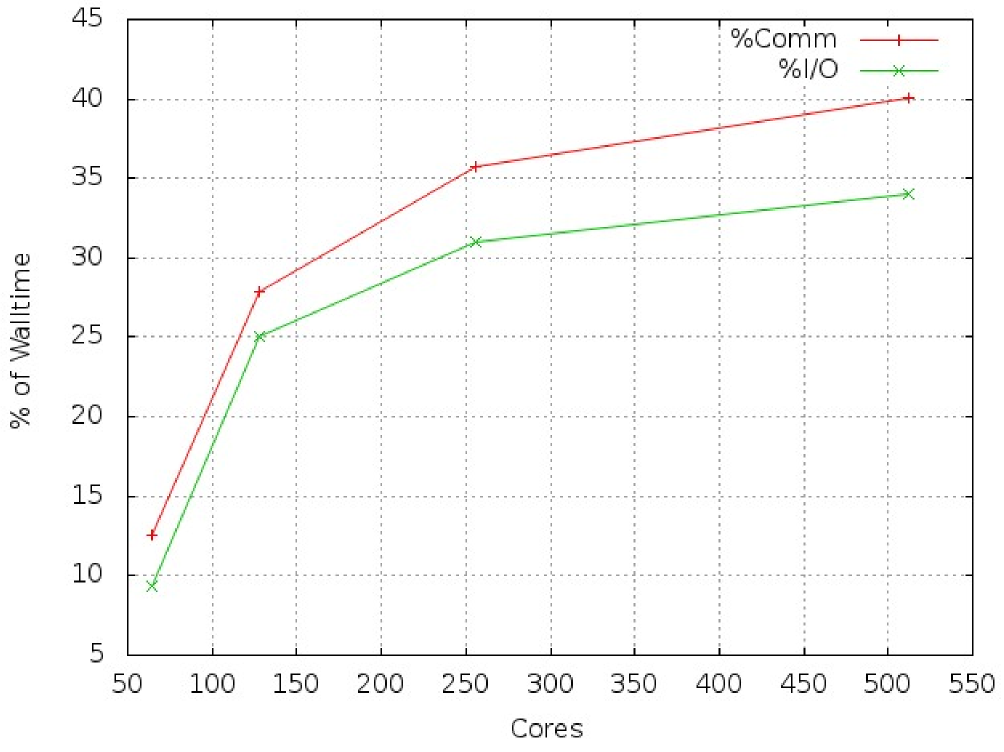


Figure 4: % of walltime taken by Communication and I/O (interFoam on Curie)

3. Model Improvements/Optimisations

Based on the results described so far, a significant amount of effort went into investigating various issues that could impinge upon performance. Our discussions for the remainder of this paper will focus on alterations that can be made to a model to ensure optimal performance. The two binaries that are run in parallel, *snappyHexMesh* and *interFoam,* will be discussed individually.

3.1 SnappyHexMesh

As stated before this tool is used for ‘snapping’ the block mesh to the surface of the body. It has been found that there are many factors affecting ithe efficiency of this method. An overall analysis has found snappyHexMesh to be a rather unreliable tool that can, in some cases, completely fail to converge. Furthermore (6) its scaling improves dramatically but subsequently drops off sharply for more than 12 cores (***Figure 5***). In this model specifically, however, such poor scaling can be quite easily understood when we consider two following factors which are discussed in the following sub-sections.

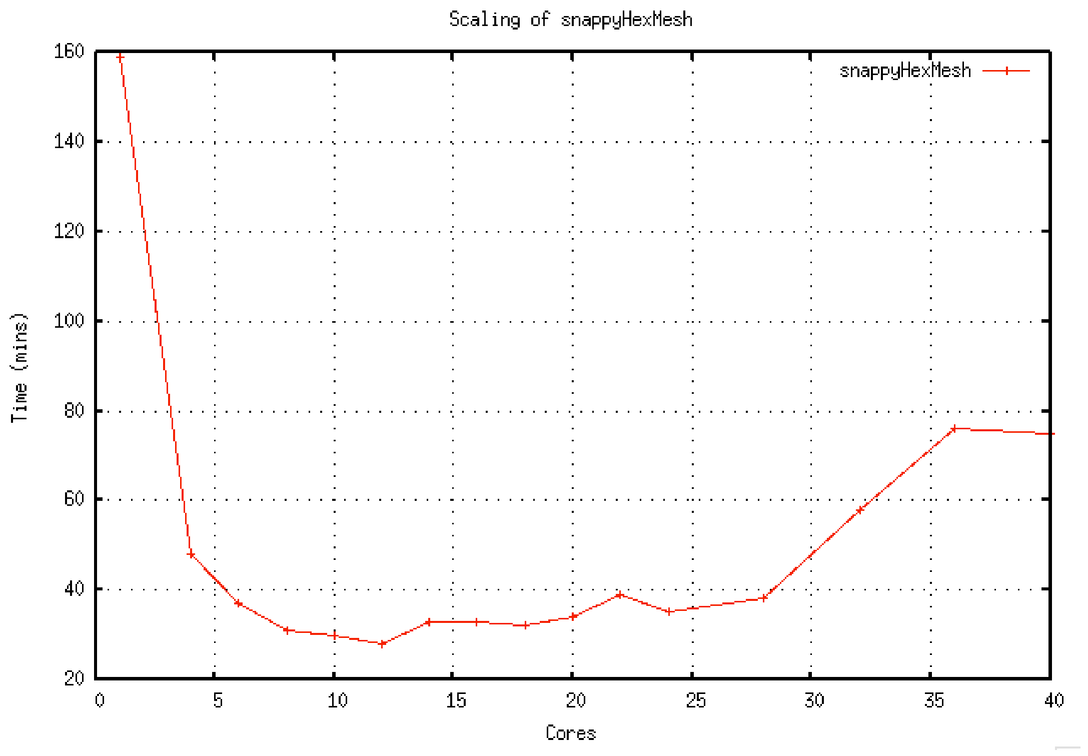


Figure 5: SnappyHexMesh Scaling

3.1.2 Refinement

SnappyHexMesh operates via a loop, known as the refinement loop, which continuously measures proximity of block cells to model cells. The *snappyHexMeshDict* contains some influential 'keyword' variables that control the behaviour of this loop. One of the most important is *minRefinementCells.* This defines the number of cells that can be 'left over' after the refinement. Of course, we can expect this to affect the overall accuracy of the simulation so it is important to make sure the model behaviour is still as expected. Furthermore, as the refinement region is reduced, there will be fewer and fewer processors involved in the 'snapping'. The outcome of this is that the majority of communication between processes is unnecessary and we would expect best performance for a lower number of cores. This could potentially lead to the case whereby *snappyHexMesh* operates best over a subset of the total number of cores, perhaps with a separate MPI communicator being implemented for *snappyHexMesh.*

3.1.3 Cell Aspect Ratio

From the definition of the block mesh (**Summary of Model***)* it is easily seen that the aspect ratio of each cell is about 2.5:1:1. By rewriting the blockMesh, keeping the same dimensions, we can alter the time required for *snappyHexMesh* to converge. ***Table 2*** shows the variation in convergence times for three different meshes.

Aspect Ratio	nCells	Nprocs	Time (min)
2.5:1:1	421,875	144	348
1:1:1	655,360	144	295
1:1:1	81,920	144	48

Table 2 snappyHexMesh Scaling

The top two meshes have the exact same dimensions but the lower of the two has an improved aspect ratio, resulting in more cells. The lowest row describes a mesh with each dimension having been halved. This immediately provides us with an interesting and important result – that the aspect ratio of the cells is more influential to scaling of *snappyHexMesh* than the number of cells in the entire domain.

These two results highlight the sensitivity of the snappyHexMesh utility to specific parameters defining the users model. This hints at the fact that each simulation in OpenFOAM must be considered individually and we cannot apply one set of general guidelines to guarantee a performance increase.

3.2 InterFoam

The other time consuming process in the execution of this model is the solver- *interFoam*. After the domain has been decomposed, each process is assigned a region over which it must solve the governing Navier-Stokes equations based on approximations defined by the solver. Assuming all regions are of equal size, each process is then responsible for the same workload, split between solver calculations, I/O operations and MPI calls. Consequently, we can assume the scaling will be dependent on a combination of these factors. The IPM profile has already shown that the scaling is not constricted by communication (MPI calls) and as we are not tampering with the existing model or algorithms we switch our attention to file I/O. The linear increases in both files read and written is shown in ***Figure 6***.

3.2.1 Solver Output

The behaviour of the solver is controlled by the entries in the *controlDict* file[d]. The most important parameter controlling output is *writeInterval*. This governs the frequency with which the solver will call on all processes to output their current state. In the existing setup of this case, the solver outputs the field states 40 times per simulation. From the Master-Slave regime employed by OpenFOAM (6) we can readily expect a communication bottleneck – whereby one process is responsible for all output. This bottleneck can be seen in the Darshan tables (***Table 1***) and corresponding graph (***Figure 6***). The graph depicts the expected linear increase of files outputted when more cores are used and the linear, although steeper, increase in files being read. Of course, one immediate solution to the issue of files being written would be to reduce or remove the write interval. However, depending on the model and output requirements of a given simulation, this may not always be a viable option. Furthermore, calculations reveal that each process consistently writes 8 files but reads between 17 and 19. This would lead us to believe that a more significant proportion of the walltime is being spent reading files rather than writing to them. Shown below in ***Table 3*** is the effect on the *interFoam* execution time by reducing the *writeInterval* by a factor of two in our simulation.

[d] http://www.openfoam.org/docs/user/controlDict.php

Output Events per execution	Total Solver Run-time
40	306mins
20	294mins

Table 3 Reduction of writeInterval

3.2.2 Solver Input

Following the less than impressive performance increases from reducing the write interval, our attention focused on file input to the solver. It was found that OpenFOAM contains the option of modifying inputs during the course of the simulation. This is either allowed or disallowed through the use of the runTimeModifiable (rTM) keyword in controlDict. Setting it to true causes the solver to re-read the contents of dictionaries at the beginning of each timestep. In the given CelticExplorer case this had been set to true and has remained so for all analysis in this paper unless otherwise stated. During the course of the simulation, however, it has been observed that none of the dictionaries are modified or altered. For this reason, runTimeModifiable has been set to false and a new set of scaling figures have been obtained. The scaling curve is shown in Figure 7 and corresponding Darshan profile. Table 4. It can be seen that Table 1 and Table 4 contain identical entries under Files Created and Files Read yet the walltime, time spent handling metadata and percentage I/O in the simulation are all drastically reduced. We can attribute this to the 100-fold decrease in the number of calls to stat(). It is not surprising that enabling such rTM modifications increases the frequency with which file timestamps are monitored. To improve performance, the user should consider their model and decide whether or not this function is required. It should only be used if necessary as it has been shown to have the most dramatic influence on the solver walltime. The user should understand the nature of this variable and the role it plays in simulations before adjusting it.

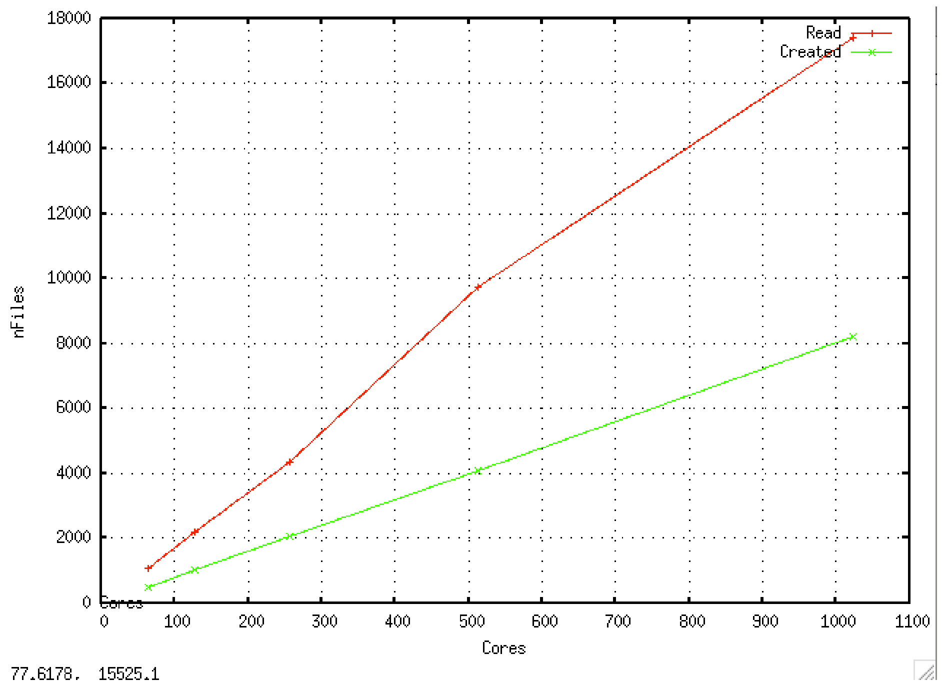
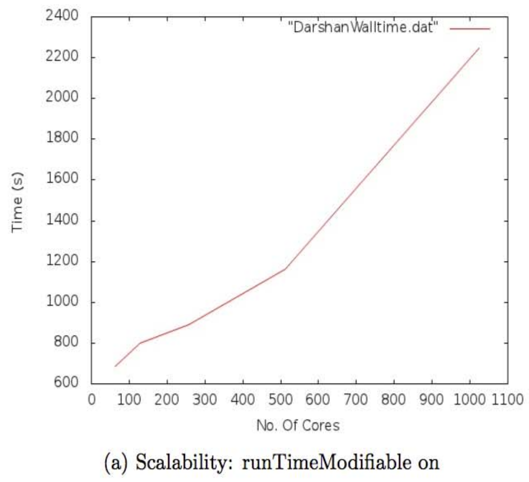
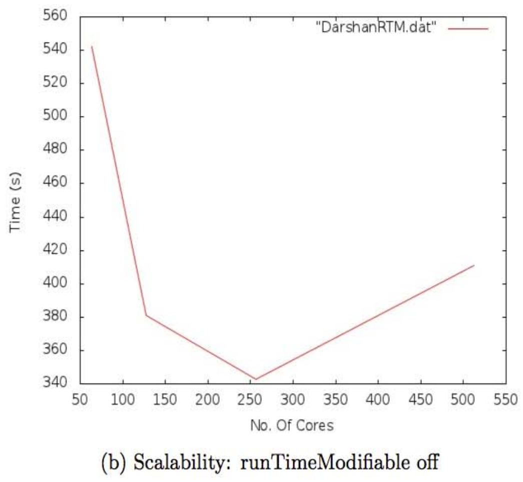


Figure 6 File I/O



(a) Scalability: runTimeModifiable on



(b) Scalability: runTimeModifiable off

Figure 7: Walltime vs Number of Cores with and without runTimeModifiable run on Curie (simulation time extended)

N Cores	Walltime (s)	Metadata Handling (s)	% I/O	Files Created	Files Read	Ave File Size (Kb)	Stat()-Calls
64	542	0.99	0.2	512	1089	597	5000
128	381	2.62	0.7	1024	2177	317	10000
256	343	6.03	1.8	2048	4353	163	2000
512	411	14.4	3.5	4096	9729	84	44000

Table 4: Darshan Profile for interFoam on Curie with runTimeModifiable = off and extended execution time

4. Conclusions

We have shown in this report that the scaling of a code the size and complexity of OpenFOAM has significant dependencies on the physics and geometry of the model involved. There are numerous factors that can affect performance and yielding optimal results relies heavily on the users understanding of, and familiarity with, their simulation and model. Indeed, what has been shown here is an application whose performance on large machines is constrained by both I/O design and communication structure, yet influenced by the physical and behavioural characteristics of the simulation. The results obtained in this study cannot necessarily be attributed solely to the design of OpenFOAM but rather to the specific case in question. Previous studies (3,5) have shown much better performance for even larger domains than the one discussed here and we do not contradict any such results. However, when combined with the results from this study it becomes clear that a fixed set of "scaling rules" does not apply to OpenFOAM. Hence, the onus is on the user to understand the geometry and physics of their model, and ensure that the measures they put in place to provide optimal performance do not alter their results or expected output.

Significantly, what has been found is that the process communication structure does not operate efficiently with run-time modifications enabled. Blocking MPI calls (MPI_Waitall, MPI_Allreduce) become dominant as file timestamps are heavily monitored through stat()-calls. The resulting communications implementation is then responsible for the dominant portion of overall walltime – up to 74% as seen in the case of 512 cores (***Table 1*** and Figure 1). However, with the runTimeModifiable parameter disabled, the volume of stat()-calls is dramatically reduced[e] (Table 4) and the walltime and scaling profile are no longer hindered by poor process communication (Figure 7 & Table 4).

From a users perspective, analyzing their model and understanding the factors that affect scaling are of the highest importance. To achieve significant petascale speed-ups, the required output and runtime behaviour of the model should be altered. A high output, interactive model will not scale well and any simulation being put forward as a candidate for Tier-0 PRACE machines should have a very lightweight communication and I/O design.

Currently within OpenFOAM process communication and file I/O are heavily linked as reading and writing to files is the chosen method of sharing data. Monitoring file timestamps with such frequency has been shown to block communication and eliminate any hope of good scaling. OpenFOAM uses a IOStream object to read and write updated parameters. IOstream is used throughout the application of more than a million lines of C++-code. A new scheme of communication and I/O are under such conditions not easily introduced. Though, it is worth investigating whether a I/O-library like parallel-NetCDF can be used by OpenFOAM.

References:

1. Darshan Profiler, Argonne National Lab, http://www.alcf.anl.gov/resource-guides/darshan
2. IPM Profiler, http://ipm-hpc.sourceforge.net
3. Pringle, Gavin; Porting OpenFOAM to HECToR, EPCC, http://www.hector.ac.uk/cse/distributedcse/reports/openfoam/openfoam.pdf
4. Lindi, Bjorn; I/O Profiling with Darshan; www.prace-ri.eu/IMG/pdf/IO-profiling_with_**Darshan**.pdf
5. OpenFOAM Website; http://www.openfoam.com/features/parallel-computing.php
6. Riviera, Orlando; Fuerlinger, Karl; Kranzlmueller, Dieter; Investigating the Scalability of OpenFOAM for the solution of Transport Equations and Large Eddie Simulations, LRZ, Munich.
7. https://bscw.zam.kfa-juelich.de/bscw/bscw.cgi/d587896/How to use IPM and Openfoam.pdf

[e] 100 times less

Appendix

Profilers

We shall describe here how to compile and link the relevant profilers to the OpenFOAM binaries. Tools such as Scalasca were attempted but ultimately proved unsuccessful and we settled on two profilers – IPM for communication profiling and Darshan for I/O profiling.

IPM and OpenFOAM

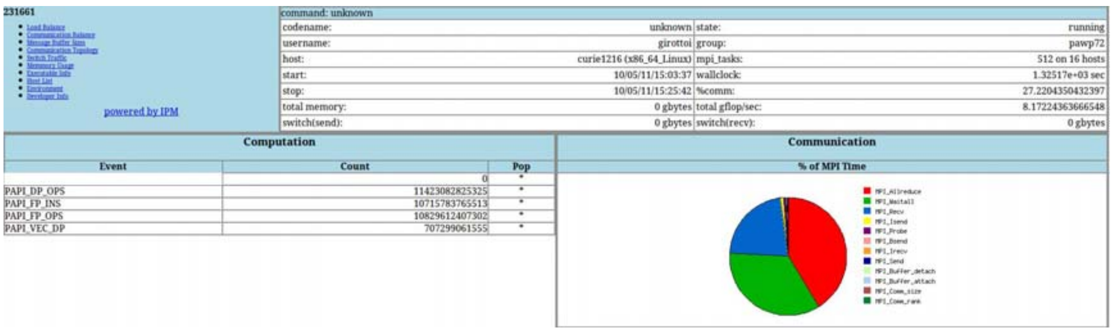
The following information is taken from a document distributed by Orlando Rivera at LRZ and altered to refer to the ICHEC case study and a version of OpenFOAM compiled on the CURIE machine (7).

- Download and build IPM according to documentation
- Assuming there exists a complete OpenFOAM-1.7.1 build
- $:vi OpenFOAMInstallPath/OpenFOAM-1.7.1/wmake/rules/linux64Icc/mplibBULLXMPI
- Edit to the following
    - PFLAGS = −DMPICH SKIP MPICXX
    - PINC = −I$(MPIARCHPATH)/include
    - PLIBS = −L$IPMHOME/lib −lipm −pthread −L$(MPIARCHPATH)/lib \−lmpi −ldl −Wl,−−export−dynamic −lnsl −lutil \ −lm −ldl
- Go to OpenFOAMInstallPath/OpenFOAM-1.7.1/src/Pstream/mpi
- $: wclean
- $: wmake libso
- Dynamically link the IPM library path ($IPM HOME/lib) to LD LIBRARY PATH
- Rebuild the solver of interest e.g. interFOAM

  $:cd applications/solvers/multiphase/interFoam

  $:wclean $:wmake libso
- Run the solver as usual.

A sample output (generated according to the documentation on the IPM website) can be seen in ***Figure 8***.



231661

command: unknown			
codename:	unknown	state:	running
username:	girottoi	group:	pawp72
host:	curie1216 (x86_64_Linux)	mpi_tasks:	512 on 16 hosts
start:	10/05/11/15:03:37	wallclock:	1.32517e+03 sec
stop:	10/05/11/15:25:42	%comm:	27.2204350432397
total memory:	0 gbytes	total gflop/sec:	8.17224363666548
switch(send):	0 gbytes	switch(recv):	0 gbytes

Computation

Event	Count	Pop
	0	*
PAPI_DP_OPS	11423082825325	*
PAPI_FP_INS	10715783765513	*
PAPI_FP_OPS	10829612407302	*
PAPI_VEC_DP	707299061555	*

Communication

% of MPI Time

Figure 8: IPM Profile

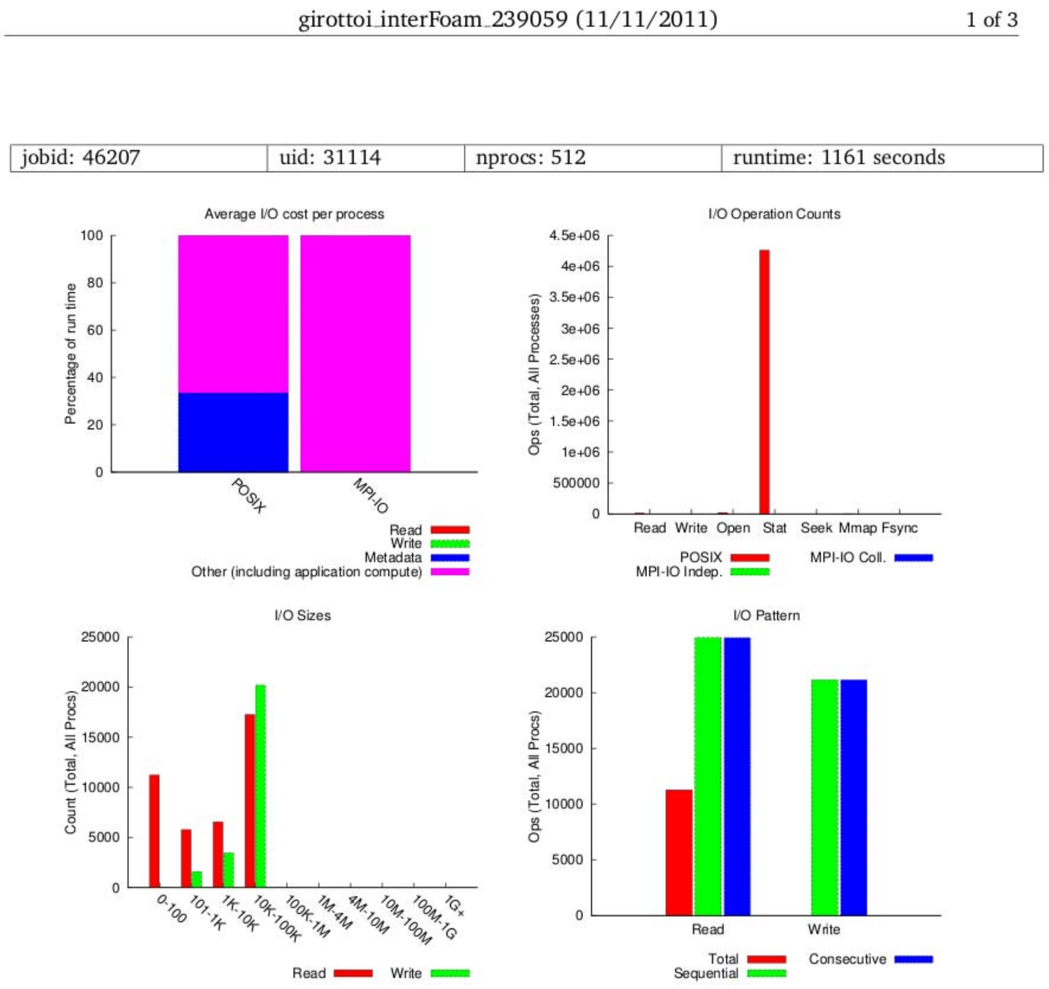
Darshan and OpenFOAM

Darshan is a petascale I/O characterisation tool designed to graphically and numerically present I/O behaviour, including patterns of access with minimum overhead (1). Having completed an IPM profile, using Darshan provides a comprehensive profile of file I/O behaviour of an application both for parallel and POSIX, and furthermore, does so with negligible overhead.

The following steps outline the methodology of linking Darshan to OpenFOAM-1.7.1

- Download and build the latest version of Darshan5 according to documentation
- Assuming there exists a complete OpenFOAM-1.7.1 build
- $:vi OpenFOAMInstallPath/OpenFOAM-1.7.1/wmake/rules/linux64Icc/mplibBULLXMPI 4 Edit to the following
    - PFLAGS = −DMPICH SKIP MPICXX
    - PINC = −I$(MPIARCHPATH)/include
    - PLIBS = −L$(DARSHANPATH)/ lib −ldarshan −pthread \−L$(MPIARCHPATH)/lib −lmpi \−ldl −Wl,−−export−dynamic −lnsl −lutil \ −lm −ldl
- Go to OpenFOAMInstallPath/OpenFOAM-1.7.1/src/Pstream/mpi
- $: wclean
- $: wmake libso
- Dynamically link the the Darshan library path ($DARSHAN PATH/lib) to LD LIBRARY PATH
- Rebuild OpenFOAM library $:cd src/OpenFOAM $:wclean
- $:wmake libso
- Run the solver as usual
- Your output is saved in a folder given to the Darshan installer e.g /ccc/scratch/cont005/pawp72/girottoi/DARSHAN LOG/ □ MONTH □ / □ DAY □ /
- Follow Darshan documentation in order to create readable output file

An example output is shown in ***Figure 9***



girottoi_interFoam_239059 (11/11/2011) 1 of 3

jobid: 46207	uid: 31114	nprocs: 512	runtime: 1161 seconds

Figure 9 Darshan Profile