Porting OpenFOAM to HECToR
A dCSE Project

Gavin J. Pringle
EPCC, The University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh, EH9 3JZ, UK

April 16, 2010

Abstract

We present the installation of OpenFOAM versions 1.6 and 1.5 on HECToR, along with guidelines for its use and how to compile up user routines. Benchmarks results are also presented., wherein it is clear that there is no simple relation between the optimum number of cores to be used for a given solver, a given simulation domain and a given number of cells. However, we can state that OpenFOAM scales well on HECToR for both simple tutorial cases and for complex industrial cases.

Contents

1 Introduction

This document forms the reports and deliverables for the Distributed CSE Support project to provide the OpenFOAM toolbox functionality to HECToR users, and to analyse its performance via benchmarking.

OpenFOAM is an open-source toolbox for computational fluid dynamics [1]. It consists of a set of generic tools to simulate complex physics for a variety of fields of interest, from fluid flows involving chemical reactions, turbulence and heat transfer, to solid dynamics, electromagnetism and the pricing of financial options. The core technology of OpenFOAM is a flexible set of modules written in C++. These are used to build a wealth of: solvers, to simulate specific problems in engineering mechanics; utilities, to perform pre- and post-processing tasks ranging from simple data manipulation to visualisation and mesh processing; and libraries, to create toolboxes that are accessible to the solvers/utilities, such as libraries of physical models. OpenFOAM has support for parallel operation, via MPI.

The goals of the project were to

- Install and test the core OpenFOAM solvers on HECToR
- Benchmark a range of the OpenFOAM modules, identified as useful via UK CFD communities
- Make OpenFOAM available to HECToR users, along with an online 'how-to' guide for using OpenFOAM on HECToR
- Disseminate the benchmark results

Ultimately the aim was to make OpenFOAM available for researchers to use on HECToR along with the relevant performance information. Ideally, this will present the machine to a new group of potential users who otherwise would not have considered it as an option, and allow them to undertake larger, more complex simulations, in a more timely fashion.

All work described in this report was undertaken by the author as part of this dCSE project, unless otherwise stated.

2 Technical Overview

This dCSE project was carried out on HECToR, the UK National Supercomputing service. Full details of the system are available on the website [2]. During the project HECToR was in its 'Phase 2a' configuration - 5664 Cray XT4 nodes, each containing an AMD 2.8GHz dual-core Opteron processor with 6GB RAM shared between the two cores, and running the CSE 2.1 operating system.

HECToR also contains an X2 Vector Unit consisting of 28 nodes with 4 vector processors per node; however, this part of the system was not used during this project.

As a results of this project, both the latest release of OpenFOAM, namely version 1.6, and version 1.5 are currently available on HECToR. Both versions are available as both dual- and quad-core versions. Currently, version 1.6.x is not available.

As part of this project, both versions have undergone basic testing to ensure correctness. Further, version 1.6 has undergone basic profiling, with a view to introducing HECToR-specific optimisations.

2.1 Licensing

OpenFOAM is open source software under the GPL.

2.2 Access

All HECToR users have access to the OpenFOAM binaries and source.

NB OpenFOAM no longer supports FoamX for versions 1.5 and older therefore, as only versions 1.5 and 1.6 are available on HECToR, FoamX has not been installed.

3 Installing OpenFOAM on HECToR

At the time of writing, all parallel queues cannot see the `/home` file system. This meant that the normal package accounts, which reside on `/home`, could not be employed for HECToR. Thus, OpenFOAM was installed in `/work/y07/y07/openfoam`.

Further, as OpenFOAM employs many of Cray's dynamic libraries, all the necessary libraries had to be copied to OpenFOAM's `/work` directory.

After some considerable effort on Cray's part [3], it was found that that default versions of many of the Cray components were unsuitable for compiling OpenFOAM. These include

- The default PGI programming environment was swapped for the GNU programming environment
- The default GNU C++ compiler was swapped for a particular version of the GNU C++ compiler, namely version 4.3.3
- The default Cray Message Passing Toolkit was swapped for a particular version of the Cray Message Passing Toolkit, namely version 3.2.0

4 How to run OpenFOAM on HECToR

This Section contains the details on how to access, compile (if necessary) and run the OpenFOAM software package on HECToR.

4.1 Running

These instructions describe how to run OpenFOAM, version 1.6, compiled for dual-core nodes, without the user modifying the source code.

NB HECToR's front-end nodes, where serial jobs are run, are dual-core and her back-end nodes, where parallel jobs are run, are quad-core. Further, all codes compiled for quad-core nodes may fail when run on dual-core nodes. Since all OpenFOAM codes can be run in either serial or parallel mode, the default, centrally installed version has been compiled for dual-core nodes.

First you must create the typical OpenFOAM working directory structure in your `work` space on HECToR, i.e., for a user with the username of `gavin`:

```
cd /work/z01/z01/gavin
mkdir OpenFOAM
cd OpenFOAM
mkdir OpenFOAM-1.6
mkdir gavin-1.6
cd OpenFOAM-1.6
mkdir etc
```

Then, copy the default OpenFOAM bashrc file from the OpenFOAM package account, `/work/y07/y07/openfoam`, into your own `etc` directory, i.e.

```
cp /work/y07/y07/openfoam/dual-core/OpenFOAM/OpenFOAM-1.6/etc/bashrc \\
   /work/z01/z01/gavin/OpenFOAM/OpenFOAM-1.6/etc/.
```

You must now edit your local copy of the OpenFOAM bashrc file, to set your User Directory. You have recently create this User Directory, called `<username>-1.6`, in your local OpenFOAM directory.

Explicitly, the following line in the file `../OpenFOAM/OpenFOAM-1.6/etc/bashrc`

```
export WM_PROJECT_USER_DIR=/work/y07/y07/openfoam/dual-core/$WM_PROJECT/
          /$USER-$WM_PROJECT_VERSION
```

must be replaced with a line which closely resembles:

```
export WM_PROJECT_USER_DIR=/work/z01/z01/gavin/$WM_PROJECT/
          /$USER-$WM_PROJECT_VERSION
```

You should now test the installation using following batch script.

```
#!/bin/bash --login
#PBS -q serial
#PBS -N testInstall
#PBS -l walltime=00:20:00
#PBS -A y07
. /opt/modules/3.1.6/init/bash
module swap PrgEnv-pgi PrgEnv-gnu
module swap gcc gcc/4.3.3
module swap xt-mpt xt-mpt/3.2.0
source /work/y07/y07/openfoam/dual-core/OpenFOAM/OpenFOAM-1.6/etc/bashrc
export LD_LIBRARY_PATH=$WM_PROJECT_DIR/mylib:$LD_LIBRARY_PATH
cd $WM_PROJECT_USER_DIR
foamInstallationTest
```

If this is configured correct, then you should now be able to run OpenFOAM as normal, i.e. from within the `<username>-1.6` directory.

4.2 Batch Scripts for the OpenFOAM tutorials

To run the tutorials, copy them from the package account into a new directory named `run` within your user directory, `<username>-1.6`, using the following the 3 commands:

```
cd $WM_PROJECT_USER_DIR
mkdir -p $FOAM_RUN
cp -r $FOAM_TUTORIALS $FOAM_RUN
```

where the third line may a few minutes to execute.

This following batch script runs tests the installation of the tutorials using HECToR's 'serial' queue.

```
#!/bin/bash --login
#PBS -q serial
#PBS -N testInstall
#PBS -l walltime=01:00:00
#PBS -A z01
. /opt/modules/3.1.6/init/bash
module swap PrgEnv-pgi PrgEnv-gnu
module swap gcc gcc/4.3.3
module swap xt-mpt xt-mpt/3.2.0
source /work/z01/z01/gavin/OpenFOAM/OpenFOAM-1.6/etc/bashrc
export LD_LIBRARY_PATH=$WM_PROJECT_DIR/mylib:$LD_LIBRARY_PATH
cd $FOAM_RUN/tutorials
Allclean
Alltest
```

NB All serial queues employ dual-core nodes thus, if users employ the quad-core version, then the results *may* not be correct.

This following batch runs the final part of the Dam Break tutorial, a parallel run of `interFOAM` on 8 cores, for user `gavin`. (The preparatory stages, namely running `blockMesh`, `setFields` and `decomposePar`, must be run before `interFoam`.).

```
#!/bin/bash --login
#PBS -l mppwidth=8
#PBS -l mppnppn=4
#PBS -N dam_tutorial
#PBS -l walltime=01:00:00
#PBS -A z01
export NSLOTS=‘qstat -f $PBS_JOBID | awk ’/mppwidth/ {print $3}’‘
export NTASK=‘qstat -f $PBS_JOBID | awk ’/mppnppn/ {print $3}’‘
. /opt/modules/3.1.6/init/bash
module swap PrgEnv-pgi PrgEnv-gnu
module swap gcc gcc/4.3.3
module swap xt-mpt xt-mpt/3.2.0
source /work/z01/z01/gavin/OpenFOAM/OpenFOAM-1.6/etc/bashrc
export LD_LIBRARY_PATH=$WM_PROJECT_DIR/mylib:$LD_LIBRARY_PATH
cd $FOAM_RUN/tutorials/multiphase/interFoam/laminar/damBreak
```

```
export MPICH_PTL_EAGER_LONG=1
aprun -n $NSLOTS -N $NTASK interFoam -parallel
```

NB All parallel jobs are run on quad-core nodes, hence we set `#PBS -l mppnppn=4`. Users can run either the dual- or quad-core versions of OpenFOAM in parallel, but the dual-core version may not perform as well.

Setting the environment variable `MPICH_PTL_EAGER_LONG=1` was found to speed up execution by around 7% for large numbers of cores, and had no adverse affect on performance for small number of cores.

4.3 Running pre- and post-processing routines on HECToR

Some pre- and post-processing codes are not parallelised and must be run in serial. Let us consider now consider `blockMesh`. To run `blockMesh` in serial on HECToR, users can employ the serial queues.

This following batch script runs `blockMesh` within an example case using HECToR's 'serial' queue.

```
#!/bin/bash --login
#PBS -q serial
#PBS -N blockMesh
#PBS -l walltime=01:00:00
#PBS -A z01
. /opt/modules/3.1.6/init/bash
module swap PrgEnv-pgi PrgEnv-gnu
module swap gcc gcc/4.3.3
module swap xt-mpt xt-mpt/3.2.0
source /work/z01/z01/gavin/OpenFOAM/OpenFOAM-1.6/etc/bashrc
export LD_LIBRARY_PATH=$WM_PROJECT_DIR/mylib:$LD_LIBRARY_PATH
cd $FOAM_RUN/example_case
blockMesh
```

NB All serial queues employ dual-core nodes thus, if users employ the quad-core version, then the results *may* not be correct.

However, it is quite possible that the amount of memory available within the serial queues is insufficient. If this is the case, then users can employ a single core on a node in the 'parallel' queues. This will furnish the user with the node's entire memory. NB this method will actually cost the price for running on 4 cores, despite only one core being used. An example batch script follows. Note the use of `aprun`, despite running the serial code, as aprun forces the code to be run in a parallel queue.

```
#!/bin/bash --login
#PBS -l mppwidth=1
#PBS -l mppnppn=1
#PBS -N blockMesh
#PBS -l walltime=01:00:00
#PBS -A z01
. /opt/modules/3.1.6/init/bash
```

```
module swap PrgEnv-pgi PrgEnv-gnu
module swap gcc gcc/4.3.3
module swap xt-mpt xt-mpt/3.2.0
source /work/z01/z01/gavin/OpenFOAM/OpenFOAM-1.6/etc/bashrc
export LD_LIBRARY_PATH=$WM_PROJECT_DIR/mylib:$LD_LIBRARY_PATH
cd $FOAM_RUN/example_case
aprun -n 1 -N 1 blockMesh
```

5 Using dual- or quad-core installations of OpenFOAM 1.6 or 1.5

At the time of writing, HECToR employed both dual- and quad-core nodes. Codes compiled for the dual-core nodes can also run on the quad-core nodes, but will not exploit the benefits of the quad-core architecture, namely the new cache hierarchy and to carry out core computational work as packed SEE instructions. However, these benefits have a down side, in that codes compiled for the quad-core nodes *may not* work correctly on dual-core nodes.

Now, at the time of writing, all codes run serially will employ dual-core nodes, whereas all codes run in parallel will employ quad-core nodes. Further, since many of OpenFOAM's solvers can be run both in parallel and in serial, both version of OpenFOAM have been installed twice, once for dual-core nodes and once for quad-core nodes.

If you require OpenFOAM 1.6 compiled for quad-core, then please employ the `OpenFOAM/quad-core/OpenFOAM-1.6` installation. If you require OpenFOAM 1.5, compiled for dual-core or quad-core, then please employ the `OpenFOAM/dual-core/OpenFOAM-1.5` and `OpenFOAM/quad-core/OpenFOAM-1.5` versions, respectively.

NB all OpenFOAM codes can be run in either serial or parallel and that codes compiled for quad-core nodes may fail when run serially, as the serial queue employs dual-core nodes only.

6 Benchmarking

This section describes the different benchmark cases employed, the methodology behind the benchmarking, and finally the results of each case.

We have investigated the following cases:

- 3D Lid-Driven Cavity Flow. Here we employ the OpenFOAM's icoFoam solver and considered two cases with 1 million and 8 million cells each.
- 2D DamBreakFine. This is the tutorial example and employs the OpenFOAM's interFoam solver.
- 3D DamBreak. This is an extension of the 2D DamBreakFine case. We employ interFoam for a 3D mesh of 6.23 million cells.
- 3D Spray. This is an extension of the 2D spray tutorial. We employ OpenFOAM's pisoFoam for 3 cases with .3 million, 2.5 million and 19 million cells.

6.1 Fruitful Connections

During the course of this project, the author met with an author of OpenFOAM [4] and established a working relationship to develop the 3D Dam Break benchmark, as described below.

Further, the author attended the Open Source CFD International Conference 2009, on the 12th and 13th of November, at the World Trade Center, Barcelona, Spain. This conference permitted the author to have indepth discussions with both academic and industrial users of OpenFOAM, which helped guide the choice of benchmarks below. These contacts included Paul Garlick, [5], who helped develop the 3D Spray benchmark, as described below.

Lastly, Patrice Calegari of Bull [6] and Esko Jarvinen of CSC [7] helped develop the 3D Lid-Driven Cavity Flow benchmark.

6.2 Methodology

6.2.1 Pre-processing

There are many pre-processing routines which must be run in serial, such as `blockMesh`. However, HECToR's 'serial' queue employs shared nodes and, as such, users only get a share of the node's total memory. During this work, it was found that, whenever the simulation was simply too large, that the 'serial' queue was too small. Thus, when this occurred, we would run the serial job in a small parallel queue. This is described in more detail, and an example batch file is presented, in Section 4.3.

However, it was found that some of the benchmark cases were simply too large to pre-process on HECToR. When this occurred, this issue was circumvented by running OpenFOAM on another platform, namely Ness at EPCC [8]. This platform has 2 nodes of 32GB each, where each node has 16 AMD Opterons. Effectively, this is a fat HECToR node, and codes such as `blockMesh`, `mapFields` and `stitchMesh`, were successful run serially in a parallel queue of 8 cores, thereby utilising 16GB of memory.

6.2.2 Timing and Performance

For each case, we present the timing and performance results in both a table and graphically in a log-log graph.

The tables all contain the wall clock time, in seconds, for that run. Also, for each time, the 'Performance' is also presented, where Performance figures are calculated simply as the time on $n/2$ cores divided by the time on n cores. Thus, a code which scales perfectly, i.e. if we were to double the cores, then we would half the execution time, the associated performance value will be 2.0. Any run with an assoicated performance figure of less than 1.4 has an efficiency less than 71% and, thus, is not considered to scale. Therefore, we may determine the optimum number of cores for a given simulation such that it is the largest number of cores where the Performance is greater than 1.4.

The figures are all plotted on a log-log graph, where each run is plotted in conjuncture with its associated perfect scaling curve.

Lastly, for all cases, the value of `writeInterval` in `controlDict` is set to be larger than total number of iterations. This ensures that no simulation snapshots are written, thereby reducing the output to be just the standard output.

6.3 3D Lid-driven Cavity Flow

This benchmark is a simple extension of the 2D lid-driven cavity flow, which is the first OpenFOAM tutorial [9].

As with the 2D case, the top wall moves in the x-direction at a speed of 1ms^{-1} whilst the remaining walls remain stationary, the flow is assumed laminar and is solved on a uniform mesh using the icoFoam solver for laminar, isothermal, incompressible flow.

Unlike the 2D case, all output was switched off. Further, two cases were create for cubic meshes with different discretisations, namely $100 \times 100 \times 100$ and $200 \times 200 \times 200$.

The $100 \times 100 \times 100$ case is of particular interest as it has been employed elsewhere [10][11] for benchmarking purposes, and is thus open for comparison.

To create the 3D case, the `blockMeshDict` file was changed to

```
hex (0 1 2 3 4 5 6 7) (100 100 100) simpleGrading (1 1 1)
```

and

```
hex (0 1 2 3 4 5 6 7) (200 200 200) simpleGrading (1 1 1)
```

For both the 100^3 and 200^3 cases, the `controlDict` file contained

```
application icoFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         0.025;
deltaT          0.005;
writeControl    timeStep;
writeInterval   20;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression uncompressed;
timeFormat      general;
timePrecision   6;
runTimeModifiable yes;
```

For 8 cores, the file `decompoiseParDict` contains the lines

```
numberOfSubdomains 8;
method          simple
simpleCoeffs
{
    n               (2 2 2);
    delta           0.001;
}
```

where

```
    n               (2 2 2);
```

denotes a cubic layout of 8 processors, $np_x = 2$, $np_y = 2$ and $np_z = 2$, in a virual grid. Indeed, many processor configurations were investigated. For instance, for 8 processors, $(8 \times 1 \times 1$, $(1 \times 8 \times 1)$, $(1 \times 1 \times 8)$, $(4 \times 2 \times 1)$, $(1 \times 2 \times 4)$, etc. Simulations were run for `numberOfSubdomains`$= 1, 2, 4, 8, ...$

It has previously been reported, [10], that varying the processor virtual topology when running a 3D lid-driven cavity flow case on a Cray XT4/XT5, has little affect on performance. Our investigation showed that, to a first approximation, this is indeed the case. However, we found a slight performance gain if the value of np_z is as low as possible. This is due to the fact that C++ stores the last dimension of arrays contiguously in memory and, if the last dimension is not distributed over processors, then more data will be kept in cache lines.

We ran the 100^3 and 200^3 cases for 5 time step, and 200^3 case again but for 40 time steps, to investigate the impact of start-up.

A summary of the results are displayed in figure 1.

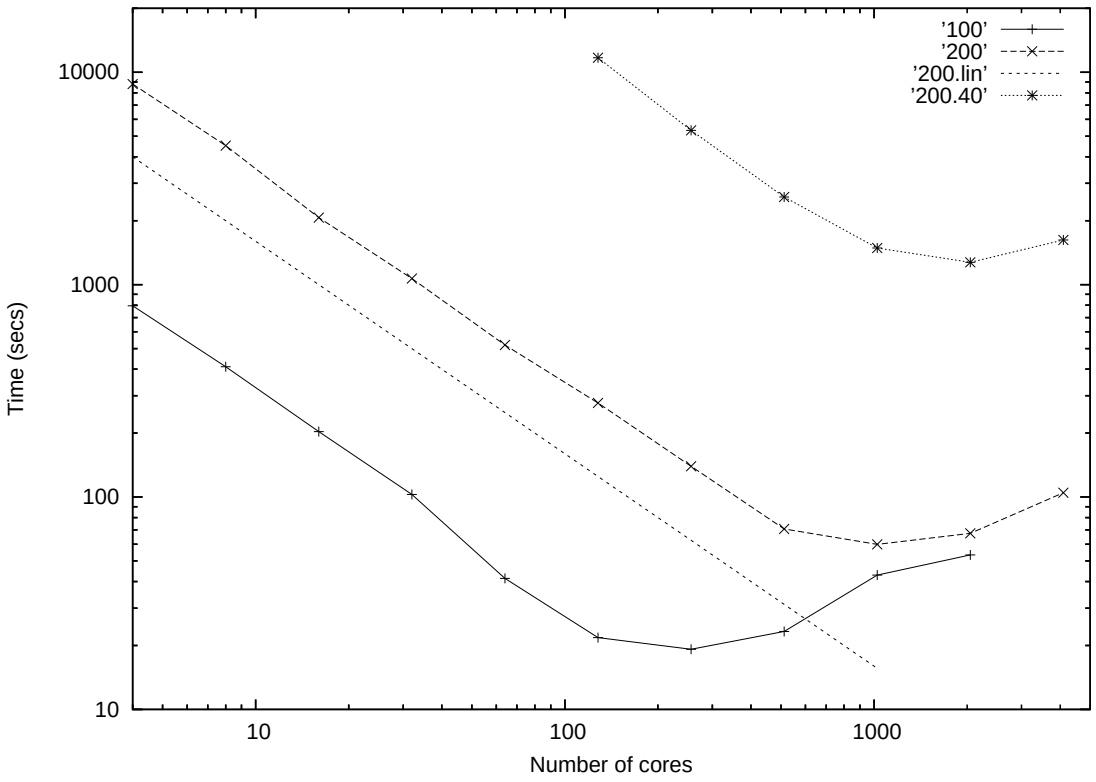

Figure 1: Timing results for 3D lid-driven cavity flow, where '100' is the 100^3 case, and '200' is the 200^3 case, both running for 5 time steps, '200.4' is the 200^3 case, both running for 40 time steps, and '200.lin' is the perfect scaling line for comparison only

The timing and performance results are presented in table 1, where performance figures are calculated as described in 6.2.2.

From table 1, it can be seen that for the 100^3 case running for 5 time steps, the optimum number of cores is 128, for the 200^3 case running for 5 time steps, the optimum number of cores is 512. However, the optimum number of cores for 200^3 case running for 40 time steps is 1024. This implies that running the simulation for only 5 time steps

Number of cores	100^3, 5 time steps Time (Performance)	200^3, 5 time steps Time (Performance)	200^3, 40 time steps Time (Performance)
4	795.3 (-)	8803.9 (-)	- (-)
8	410.7 (1.94)	4514.0 (1.95)	- (-)
16	203.1 (2.02)	2069.0 (2.18)	- (-)
32	102.9 (1.97)	1068.9 (1.94)	- (-)
64	41.4 (2.49)	519.8 (2.06)	- (-)
128	**21.8 (1.90)**	277.2 (1.88)	11691.6 (-)
256	19.2 (1.14)	139.5 (1.99)	5322.1 (2.20)
512	23.3 (0.82)	**70.7 (1.97)**	2586.1 (2.06)
1024	42.9 (0.54)	59.8 (1.18)	**1488.7 (1.73)**
2048	53.4 (0.80)	67.4 (0.78)	1272.6 (1.17)
4096	- (-)	104.7 (0.73)	1623.9 (0.78)

Table 1: Timing and performance results for 3D lid-driven cavity flow

was not long enough to remove the startup cost of the simulation.

Thus, from this exercise, we can suggest that, if using icoFoam in 3D, with a regular mesh, then for simulations with 200^3, or 8 million grid points, we recommend running on 1024 cores.

6.3.1 Environment Variables

At this point, two environment variables were then tested, namely `MPICH_PTL_EAGER_LONG` and `MPICH_PTL_MATCH_OFF`.

The environment variable `MPICH_PTL_EAGER_LONG=1`, uses the `EAGER` path for all messages, regardless of their size.[1]

The environment variable `MPICH_PTL_MATCH_OFF=1` is useful for simulations which have many small messages and is latency bound.

The '200.40' experiment (200^3 case running for 40 time steps) was run three more times, once with `MPICH_PTL_MATCH_OFF` set, once with `MPICH_PTL_EAGER_LONG`, and once with both environment variables set.

The actual timing result are not contained in this report; however, it was found that neither environment variables had an adverse effect on performance, irrespective of the process count. For large processor counts, it was found that setting `MPICH_PTL_EAGER_LONG` could give up to a 7% speed-up, especially around the optimum number of processors. Thus, we employed `MPICH_PTL_EAGER_LONG=1` for all other benchmarking in this report, and include it in the example batch scripts4.2.

Both the 100^3 and the 200^3 benchmark cases are available on the HECToR's OpenFOAM web page [12].

[1]The default is normally dependent on the total number of MPI ranks, npes say, and is normally set to $1000000 \times 2048/$ npes, with a minimum value of 1024 bytes and a maximum value of 128,000 bytes.

6.4 Dam Break Tutorial

The 2D Dam Break tutorial has been tested, using the 2nd part of the tutorial, namely the `multiphase/interFoam/laminar/damBreakFine`.

Following the tutorial, but also employing `MPICH_PTL_EAGER_LONG`, the simulation was run for `numberOfSubdomains`$= 1, 2, 4, 9 \text{and} 16$, and the resultant times are displayed graphically in figure 2 and presented in table 2.

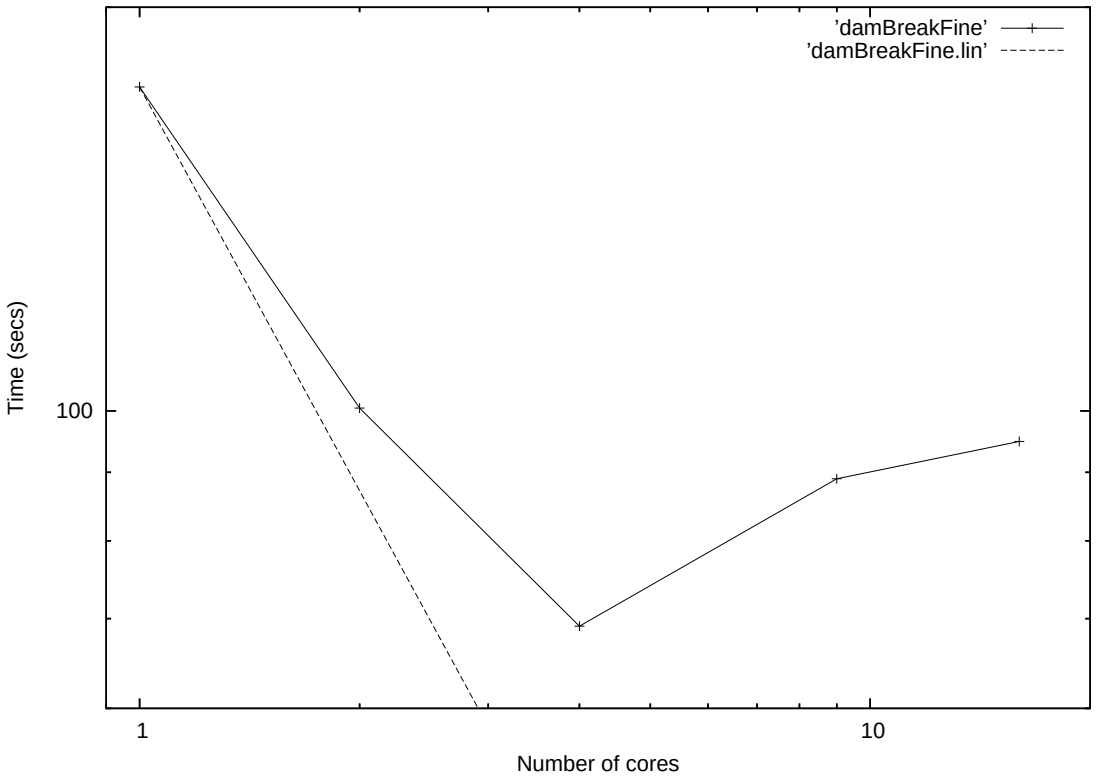

Figure 2: Time (secs) for the Dam Break Fine tutorial, where 'damBreakFine' is the time to run the tutorial and 'damBreakFine.lin' is the line of perfect scaling

Number of cores	damBreakFine Time (Perf)
1	174.4 (-)
2	100.5 (1.74)
4	**69.1 (1.45)**
9	89.0 (0.77)
16	94.9 (0.94)

Table 2: Timing and performance results for the 2D Dam Break Fine tutorial

It can be seen from table 2 that the optimum number of processors is 4; moreover, it is clear from figure 2 that this problem is simply too small to warrant running in production on HECToR. The results are included here for the sake of comparison, as this case is run often as part of the OpenFOAM tutorial.

6.5 3D Dam Break

With the assistance of OpenCFD [4], we then extended the 2D Dam Break tutorial case to run in full 3D, with such a fine mesh that water droplets can be seen.

The value of hex in `blockMeshDict` is set to

```
hex (0 1 2 3 4 5 6 7) (184 184 184) simpleGrading (1 1 1)
```

which gives 6.23 million cells.

Interestingly, as part of the pre-processing routines, the routine `blockMesh` failed due to lack of memory when running in HECToR's serial queue. This was circumvented by running `blockMesh` within a small parallel queue. (See Section 4.3).

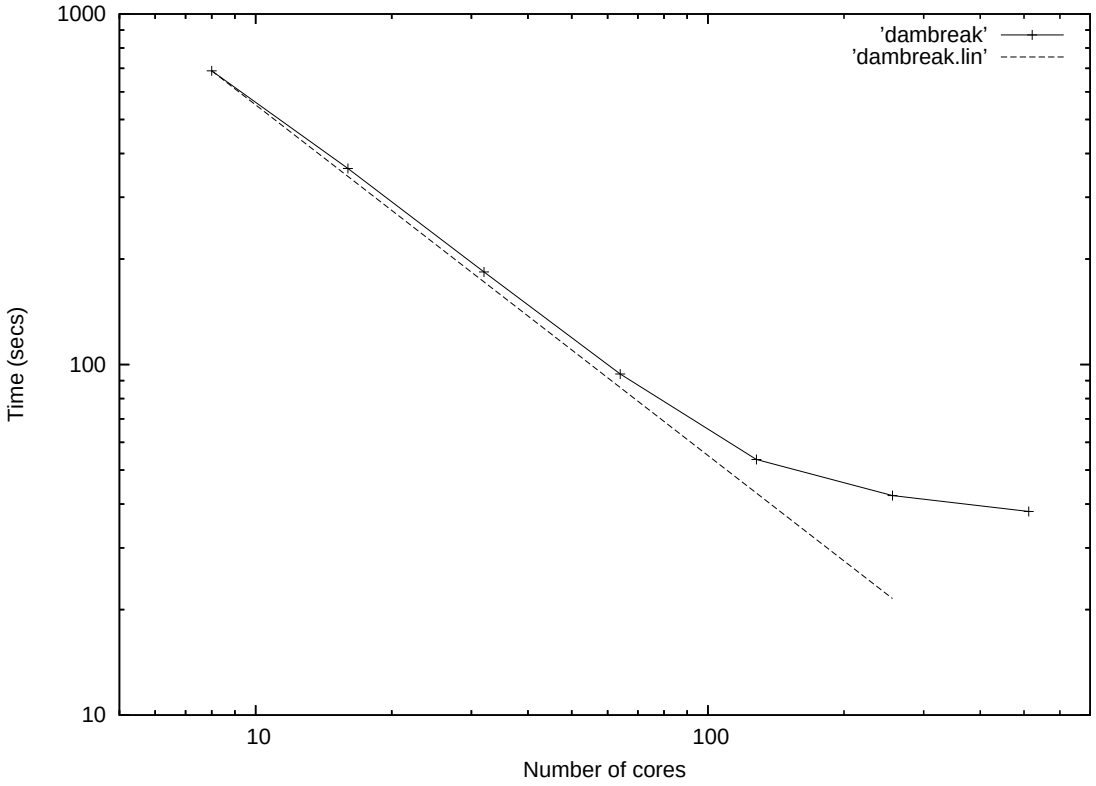

Figure 3: Time (secs) for the 3D Dam Break case, where 'damBreak' is the execution times and 'damBreak.lin' is the line of perfect scaling

It can be seen from table 3 that the optimum number of processors is 128, when running interFoam using 6.23 million cells.

The case was configured so that increasing the resolution was simply updating the value of hex in `blockMeshDict` to read

```
hex (0 1 2 3 4 5 6 7) (230 230 230) simpleGrading (1 1 1)
```

which gives 12.17 million cells. However, it was found that this case was simply too large to pre-process on either HECToR or Ness.

The 3D Dam Break benchmark case is available on the HECToR's OpenFOAM web page [12].

Number of cores	3D Dam Break Time (Perf)	processor topology
8	688.3 (-)	$4 \times 2 \times 1$
16	362.4 (1.90)	$4 \times 4 \times 1$
32	183.8 (1.97)	$4 \times 4 \times 2$
64	94.0 (1.96)	$4 \times 4 \times 4$
128	**53.6 (1.75)**	$8 \times 4 \times 4$
256	42.3 (1.27)	$8 \times 8 \times 4$
512	38.1 (1.11)	$8 \times 8 \times 8$

Table 3: Timing,and performance results and processor topology employed or 3D Dam Break case with 8.23 million cells

6.6 3D Spray

For this final timing exercise, we wished to not only employ another of the many OpenFOAM codes but also to model a real world simulation of industrial-scale. To this end, the author and Paul Garlick [5] worked closely to generate and run a 3D turbulent jet Large Eddy Simulation (LES).

The 3D model is a co-axial atomization model, where a central liquid jet is broken into droplets by a high-speed surrounding air jet. Applications of this technology are spray coating equipment and fuel combustors (for jet engines and gas turbines). This 3D case was based on the primary atomization example case in OpenFOAM, which itself is a 2D LES.

Initially, a relatively simple grid was developed for a typical geometry of a co-axial atomizer, whilst considering how to specify realistic boundary conditions to give a jet break-up length and drop sizes that can be predicted and compared to past experiment and analysis.

Then the 3D simulation was run using a Reynolds-Averaged Navier-Stokes (RANS) model, on a relatively coarse grid. The output was a steady-state solution which yielded information of the turbulence length scales. The minimum length scale was then used to define the cell size for the more detailed LES model. Finally, the results from the RANS model was used to initialize the LES model.

The image of the geometry for the atomization model is given in figure 4.

Figure 4, as generated by Paul Garlick [5], shows the inlet locations and boundary meshes. The liquid enters the domain through the central (blue) patch; the atomizing air enters through the annular (red) patch. This figurer uses a $1 \times 1 \times 1$cm cube, and a mesh with `elementSize`=0.2mm. This sets the size of the elements at the inner edge of the air inlet patch. The element size reduces towards the centre-line, where the atomization occurs, and increases towards the outer boundaries.

For this benchmark, we have 3 different cases, namely jet_coarse, jet_medium and jet_fine, and are summarised in table 4.

In table 4 we can see the value for the time step, namely `deltaT`, is smaller for the fine case than for both the coarse and medium cases. This is because the maximum Courant number increases for the finer mesh, and this lower value ensures the the maximum Courant number will be 0.1.

The $2 \times 2 \times N$ decomposition should work well for small values of N. The Scotch

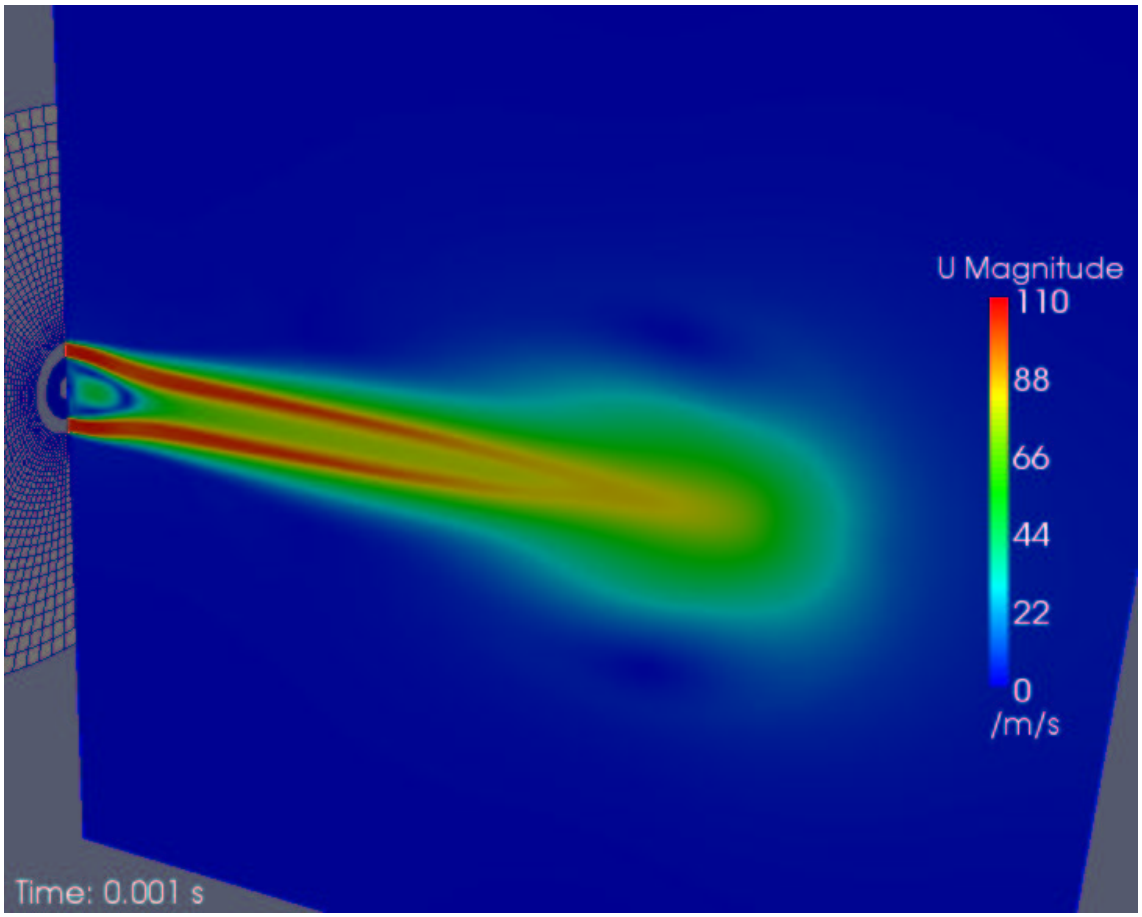

Figure 4: 3D Jet [5]

Method may be more efficient for large N.

The file `controlDict` contains the following lines.

```
application     pisoFoam;
startFrom       latestTime;
startTime       0;
stopAt          endTime;
endTime         0.00201;
deltaT          5.0e-08;
writeControl    timeStep;
writeInterval   200000;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression compressed;
timeFormat      general;
timePrecision   6;
runTimeModifiable yes;
```

Grid	`elementSize`	No. of cells	Reynolds No.	`deltaT` (secs)
coarse	0.2 mm	300,000	12,000	5.0e-08
medium	0.1 mm	2,500,000	12,000	5.0e-08
fine	0.05 mm	19,000,000	12,000	2.5e-08

Table 4: Summary of the 3 Jet Cases

6.6.1 Benchmarking the parallel versions

Firstly, some of the pre-processing was to large for HECToR, even when ran in a small parallel queue, thus most pre-processing was performed on Ness.

The method of decomposition was to set the virtual process topology using $np_x = 2$, $np_y = 2$ and $np_z = 1, 2, 4, ...$, using a simple decompositition method. This ensured that we had a balanced load-balance for the x-y plane at least.

The timing results for the 3 cases are presented graphically in figure 5 and presented in table 5, where can see that the code scales well for all 3 cases, where the optimum number of cores is 64, 256 and 512 for the coarse, medium and fine cases, respectively.

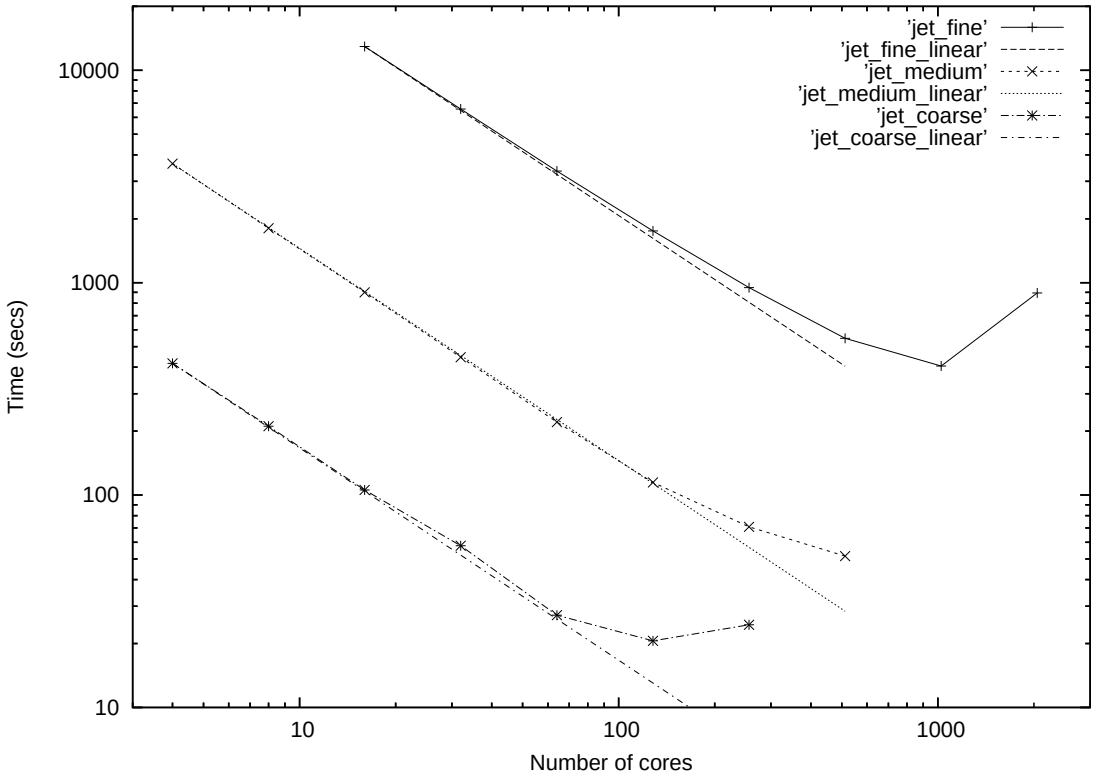

Figure 5: Time (secs) for the three 3D Jet Break Up cases, where jet_fine, jet_medium and jet_coarse are the times for the fine, medium and coarse cases, and jet_fine_linear, jet_medium_linear and jet_coarse_linear, are their respective perfect scaling curves.

Number of cores	Coarse Mesh Time (Perf)	Regular Mesh Time (Perf)	Fine Mesh Time (Perf)
4	417.0 (-)	3630.5 (1.18)	- (-)
8	211.1 (1.98)	1804.6 (2.01)	- (-)
16	105.6 (2.00)	900.9 (2.00)	12931.4 (-)
32	57.7 (1.83)	445.4 (2.02)	6568.7 (1.97)
64	**27.2 (2.12)**	220.5 (2.02)	3354.1 (1.96)
128	20.6 (1.32)	114.7 (1.92)	1752.0 (1.91)
256	24.5 (0.84)	**70.9 (1.62)**	947.6 (1.85)
512	- (-)	51.6 (1.37)	**546.8 (1.73)**
1024	- (-)	- (-)	404.9 (1.35)
2048	- (-)	- (-)	894.9 (0.45)

Table 5: Timing and performance results for 3D Jet Break Up

7 Conclusion

The main thrust of this work was to port OpenFOAM to HECToR and to give users directions on how to run their simulations efficiently. To this end, we have benchmarked a number of cases to give an idea of how many cores should be employed for a given number of mesh points and a given solver.

The results of the benchmarking is summarised in tabe 6.

Case	name	solver	No. of cells	Optimum no. of cores
3D cavity	'100'	icoFoam	1,000,000	256
3D cavity	'200.40'	icoFoam	8,000,000	1024
3D dam break	'dambreak'	interFoam	6,230,000	128
3D jet spray	'jet_coarse'	pisoFoam	300,000	64
3D jet spray	'jet_medium'	pisoFoam	2,500,000	256
3D jet spray	'jet_fine'	pisoFoam	19,000,000	512

Table 6: Benchmarking summary

It is clear from table 6 that there is no simply relation between the optimum number of cores to be used for a given solver and a given number of cells. However, we can state that OpenFOAM scales well on HECToR for both simple tutorial cases and for complex industrial cases.

OpenFOAM is proving to be a very popular CFD package. This is, in part, due to its open-source nature, wherein, unlike commercial CFD packages, users can examine the code and alter it as required (although code written by users is typically not included in future OpenFOAM releases). Further, the code scales well and does not require a per-core license one finds with commercial codes which can be financially prohibitive.

7.1 Future Work

The current version of OpenFOAM, namely 1.6.x, contains many bug fixes, and this version should be maintained on HECToR. However, it is not clear when a new version of 1.6.x is released, as there is no clear centralised announcement service, thus it may

be prudent to `git` the current version on the 1st of each month and/or after HECToR undergoes an upgrade.

7.2 Acknowledgements

Thanks are due to Chris Greenshields of OpenCFD, Paul Garlick of Tourbillion Technology Ltd, Patrice Calegari of Bull and Esko Jarvinen of CSC.

A Editing and compiling the OpenFOAM source code

To modify the source code you must copy the gzipped tarball from the OpenFOAM package account and install it in your local work directory. NB this gzipped tarball is an HECToR-specific version of OpenFOAM-1.6, in that the tarball contains additional HECToR-specific dynamic libraries and example batch scripts, and excludes html documentaion, and some unrequired third-party packages (malloc, OpenMPI, zlib and gcc).

In your designated work directory, copy and unpack the gzipped tarball into a new directory named OpenFOAM, i.e.

```
cd /work/z01/z01/gavin
mkdir OpenFOAM
cd OpenFOAM
cp /usr/local/packages/openfoam/OpenFOAM-1.6-HECToR.tar.gz .
gunzip OpenFOAM-1.6-HECToR.tar.gz
tar xvf OpenFOAM-1.6-HECToR.tar
```

This will create the 'OpenFOAM-1.6' and 'ThirdParty-1.6' directories in your workspace.

To compile, update the `OpenFOAM-1.6/etc/bashrc` file to point to your new local installation directory, i.e. for user `gavin`, change

```
 foamInstall=/work/y07/y07/openfoam/dual-core/OpenFOAM
```

to

```
 foamInstall=/work/z01/z01/gavin/OpenFOAM
```

and

```
export WM_PROJECT_USER_DIR=/work/y07/y07/openfoam/dual-core/$WM_PROJECT/
      $USER-$WM_PROJECT_VERSION
```

to

```
export WM_PROJECT_USER_DIR=/work/z01/z01/gavin/$WM_PROJECT/
      $USER-$WM_PROJECT_VERSION
```

Then, to compile, update and submit the batch script `OpenFOAM-1.6/compile_OF`, i.e.,

```
#!/bin/bash --login
#PBS -q serial
#PBS -N compile\_OF
#PBS -l walltime=05:00:00
#PBS -A y07
. /opt/modules/3.1.6/init/bash
module swap PrgEnv-pgi PrgEnv-gnu
module swap gcc gcc/4.3.3
module swap xt-mpt xt-mpt/3.2.0
source /work/y07/y07/openfoam/dual-core/OpenFOAM/OpenFOAM-1.6/etc/bashrc
export LD_LIBRARY_PATH=$WM_PROJECT_DIR/mylib:$LD_LIBRARY_PATH
```

```
cd $WM_PROJECT_DIR
cp wmake/rules/crayxt/general.orig wmake/rules/crayxt/general
./Allwmake >& make.out.1
cp wmake/rules/crayxt/general.temp wmake/rules/crayxt/general
./Allwmake >& make.out.2
cp wmake/rules/crayxt/general.orig wmake/rules/crayxt/general
./Allwmake >& make.out.3
```

NB The standard output will appear in three files, namely make.out.1, make.out.2 and make.out.3. This is because, under the current OS on HECToR, namely CSE 2.1, the default compilation will fail, however, this behaviour is expected. This is avoided by performing 3 different phases of compilation, where the standard output of each phase is written to make.out.1, make.out.2 and make.out.3. This complication will be resolved once HECToR starts running CSE 2.2.)

B Changes to Source code for HECToR

B.1 Overview

This page describes, in detail, the changes that we made to the OpenFOAM 1.6 source code, for 'Linux 64bit', in order to port the code to HECToR.

B.2 Changes to `etc/bash`

The line

```
: ${WM_COMPILER:=Gcc}; export WM_COMPILER
```

was changed to

```
: ${WM_COMPILER:=}; export WM_COMPILER
```

and

```
# WM_MPLIB = | OPENMPI | MPICH | MPICH-GM | HPMPI | GAMMA | MPI | QSMPI
: ${WM_MPLIB:=OPENMPI}; export WM_MPLIB
```

was changed to

```
: ${WM_MPLIB:=MPT}; export WM_MPLIB
```

despite MPT not being an option.

We replaced

```
WM\_ARCH=linux64
export WM\_COMPILER\_LIB\_ARCH=64
export WM\_CC='gcc'
export WM\_CXX='g++'
export WM\_CFLAGS='-m64 -fPIC'
export WM\_CXXFLAGS='-m64 -fPIC'
export WM\_LDFLAGS='-m64'
```

with

```
if [ -e /proc/cray\_xt ]; then
  WM\_ARCH=crayxt
else
  WM\_ARCH=linux64
  export WM\_COMPILER\_LIB\_ARCH=64
  export WM\_CC='gcc'
  export WM\_CXX='g++'
  export WM\_CFLAGS='-m64 -fPIC'
  export WM\_CXXFLAGS='-m64 -fPIC'
  export WM\_LDFLAGS='-m64'
fi
```

We changed the lines

```
# \_foamSource $WM_PROJECT_DIR/etc/apps/paraview/bashrc
\_foamSource $WM_PROJECT_DIR/etc/apps/paraview3/bashrc
\_foamSource $WM_PROJECT_DIR/etc/apps/ensight/bashrc
```

to

```
# \_foamSource $WM_PROJECT_DIR/etc/apps/paraview/bashrc
# \_foamSource $WM_PROJECT_DIR/etc/apps/paraview3/bashrc
\_foamSource $WM_PROJECT_DIR/etc/apps/ensight/bashrc
```

i.e. simply commenting out the paraview3 line.
Finally, the line

```
# foamInstall=$HOME/$WM_PROJECT
```

become

```
foamInstall=/work/y07/y07/openfoam/dual-core/$WM_PROJECT
```

B.3 Changes to `settings.sh`

We added the following lines

```
# compilerInstall=OpenFOAM
compilerInstall=System
case "${compilerInstall:-OpenFOAM}" in
System)
    if [ -e /proc/cray\_xt ]; then
      export WM_COMPILER_DIR=$GCC_PATH
      export WM_COMPILER_BIN=$WM_COMPILER_DIR/bin
      export WM_COMPILER_LIB=$WM_COMPILER_DIR/snos/lib64
    fi
    ;;
```

and

```
MPT)
    export MPICH_PATH=$MPICHBASEDIR
    export MPI_ARCH_PATH=$MPICH_DIR
    export MPICH_ROOT=$MPI_ARCH_PATH

    _foamAddLib $MPI_ARCH_PATH/lib
    _foamAddLib $MPICH_PATH/pmi/lib
    _foamAddLib $MPICH_PATH/util/lib

    export FOAM_MPI_LIBBIN=$FOAM_LIBBIN/mpt
    ;;
```

B.4 Changes to `wmake/rules`

We added a new `crayxt` directory, based on the contents of the `linx64Gcc` directory.

The file `c++` is identical, except

```
c++WARN = -Wall -Wno-strict-aliasing -Wextra -Wno-unused-parameter //
          -Wold-style-cast -Wnon-virtual-dtor
```

becomes

```
c++WARN = -Wall -Wno-strict-aliasing -Wextra -Wno-unused-parameter //
          -Wold-style-cast
```

The file `cOpt` is identical, except

```
cOPT = -O3 -fno-gcse
```

becomes

```
cOPT = -O3
```

The file `general` is identical, except

```
LD         = ld
```

becomes

```
LD         = ld -A64
```

Further, we added two new files named `general.orig` and `general.temp`, where the latter is a copy of the updated `general` file, and the former has replaced

```
PROJECT_LIBS = -l$(WM_PROJECT) -Wl,--whole-archive -L$(SE_DIR)/lib/snos64 //
               -lportals -Wl,--no-whole-archive -liberty -ldl
```

with

```
 PROJECT_LIBS = -l$(WM_PROJECT) -liberty -ldl
```

Using three files in this manner makes the compilation much smoother (see below).

The file `X` is changed from

```
XINC       = $(XFLAGS) -I/usr/include/X11
XLIBS      = -L/usr/lib64 -lXext -lX11
```

to

```
XINC       = $(XFLAGS) -I/usr/X11R6/include
XLIBS      = -L/usr/X11R6/lib64 -lXext -lX11
```

Finally, three new files were added: two executable binaries, namely `dirToString` and `wmkdep`, and the text file `mplibMPT`, which describes the locations of the required parts of Cray's MPI library. Specifically, the file contains the following lines

```
PFLAGS =
PINC = -I$(MPI_ARCH_PATH)/include
PLIBS = -L$(MPI_ARCH_PATH)/lib -L$(MPICH_PATH)/pmi/lib //
        -L$(MPICH_PATH)/util/lib -lmpich -lpmi -lalpslli //
        -lalpsutil -rt
```

B.5 Changes to Pstream

We added an MPT environment variable to `src/Pstream/Allwmake` to employ the Cray MPI library. Specifically,

```
case "$WM_MPLIB" in
*MPI*)
```

was changed to

```
case "$WM_MPLIB" in
MPT | *MPI*)
```

B.6 Changes to parMetis

The file `src/decompositionAgglomeration/parMetisDecomp/Make/options` was updated from

```
EXE_INC = \
    $(PFLAGS) $(PINC) \
    -I$(WM_THIRD_PARTY_DIR)/ParMetis-3.1/ParMETISLib \
    -I$(WM_THIRD_PARTY_DIR)/ParMetis-3.1 \
    -I../decompositionMethods/lnInclude
```

to read

```
EXE_INC = \
    $(PFLAGS) $(PINC) \
    -DMPICH_IGNORE_CXX_SEEK \
    -I$(WM_THIRD_PARTY_DIR)/ParMetis-3.1/ParMETISLib \
    -I$(WM_THIRD_PARTY_DIR)/ParMetis-3.1 \
    -I../decompositionMethods/lnInclude
```

B.7 More on `mylib`

Here is the complete list of libraries contained in the `mylib` directory, where some are copied from the front end and some are simply dummy stubs to satisfy runtime requirements.

```
-rwxr-xr-x  1 gavin z01    8179 Mar 30 21:06 libportals.so.1
-rwxr-xr-x  1 gavin z01  201330 Jun  1 14:32 libpmi.so
-rwxr-xr-x  1 gavin z01   37643 Jun  1 14:34 libalpsutil.so.0.0.0
-rwxr-xr-x  1 gavin z01   37643 Jun  1 14:34 libalpsutil.so.0
-rwxr-xr-x  1 gavin z01   37643 Jun  1 14:34 libalpsutil.so
-rwxr-xr-x  1 gavin z01   56080 Jun  1 14:34 libalpsutil.a
-rwxr-xr-x  1 gavin z01   33215 Jun  1 14:34 libalpslli.so.0.0.0
-rwxr-xr-x  1 gavin z01   33215 Jun  1 14:34 libalpslli.so.0
-rwxr-xr-x  1 gavin z01   33215 Jun  1 14:34 libalpslli.so
-rwxr-xr-x  1 gavin z01   46908 Jun  1 14:34 libalpslli.a
-rwxr-xr-x  1 gavin z01 1512997 Jun  1 14:36 libmpich.so.1.1
-rwxr-xr-x  1 gavin z01 1505121 Jun  1 14:39 libc.so.6
-rwxr-xr-x  1 gavin z01   49207 Jun  1 14:39 librt.so.1
-rwxr-xr-x  1 gavin z01  114562 Jun  1 14:40 libpthread.so.0
-rwxr-xr-x  1 gavin z01  404881 Jun  1 14:40 libm.so.6
-rwxr-xr-x  1 gavin z01   19808 Jun  1 14:40 libdl.so.2
-rwxr-xr-x  1 gavin z01  478196 Jun  1 14:43 libgcc_s.so.1
-rwxr-xr-x  1 gavin z01 5118313 Jun  1 14:43 libstdc++.so.6
```

B.8 Compilation

We employed the following serial batch script to compile

```
#!/bin/bash --login
#PBS -q serial
#PBS -N compile_OF
#PBS -l walltime=05:00:00
#PBS -A y07
cat $0
. /opt/modules/3.1.6/init/bash
module swap PrgEnv-pgi PrgEnv-gnu
#following line makes serial codes may break on front end dual core nodes.
#module load xtpe-barcelona
module swap gcc gcc/4.3.3
module swap xt-mpt xt-mpt/3.2.0
cd /work/y07/y07/openfoam/dual-core/OpenFOAM/OpenFOAM-1.5
source etc/bashrc
export LD_LIBRARY_PATH=/work/y07/y07/openfoam/dual-core/OpenFOAM/
                       OpenFOAM-1.5/mylib:$LD_LIBRARY_PATH
./Allwmake >& make.out
```

This fails after around four hours with

```
make[3]: *** [../OpenFOAM/OpenFOAM-1.5/lib/crayxtDPOpt/libuserd-foam.so] Error 1
```

This is resolved by editing

```
../OpenFOAM/OpenFOAM-1.5/wmake/rules/crayxt/general
```

and replacing

```
PROJECT_LIBS = -l$(WM_PROJECT) -Wl,--whole-archive -L$(SE_DIR)/lib/snos64 //
               -lportals -Wl,--no-whole-archive -liberty -ldl
```

with

```
PROJECT_LIBS = -l$(WM_PROJECT) -liberty -ldl
```

then remaking. We then reinstate this edited line in general and remake once more.

This workaround is necessary due to the OS Cray XT4 CSE 2.1 (as it is now) does not ship the necessary dynamic libraries. This will be fixed in CSE 2.2.

Note, that there are some 'portals' warnings (Cray builds its MPI using 'portals'), but we can ignore them, i.e. they appear as `mylib is not in LD_LIBRARY_PATH`. They are, however, not required as we are link in the static portals code. There are a number of dummy libraries in mylib is just to satisfy runtime requirements, including a dummy portal library stub.

References

[1] OpenFOAM web page, http://www.opencfd.co.uk/openfoam OpenFOAM web page

[2] HECToR website, http://www.hector.ac.uk

[3] Beech-Brandt, J., Cray Centre of Excellence, 2009, pers comm.

[4] Greenshields, C., OpenCFD, 2010, pers comm.

[5] Garlick, P., Tourbillion Technology Ltd, http://www.tourbillion-technology.com, 2010, pers, comm.

[6] Calegari, P., Bull, 2010, pers comm.

[7] Jarvinen, E., CSC, 2010, pers comm.

[8] Ness User Guide: http://www2.epcc.ed.ac.uk/∼ness/documentation/index.html

[9] OpenFOAM User Documentation, http://www.opencfd.co.uk/openfoam/doc/user.html

[10] OpenFOAM Performance on Cray XT4/XT5, Esko Jarvinen, OpenFOAM-foorumi, http://openfoamfoorumi.com/wordpress/wp-content/uploads/2009/02/ofperformanceoncray_foorumi.pdf, 2009.

[11] Calegari, P., Gardner, K., Loewe, B., Performance Study of OpenFOAM v1.6 on a Bullx HPC Cluster with a Panasas Parallel File System, Open Source CFD conference, Barcelona, Nov., 2009.

[12] HECToR OpenFOAM pages: https://wiki.hector.ac.uk/userwiki/OpenFOAM

[13] Garlick, P. and Jusaja, A. K., Laser Sheet Imaging of High-Velocity Air Atomised Water Sprays, 11th International Conference on Liquid Atomization and Spray Systems, 2009.