DCSE Supporting Build of OpenIFS on HECToR

Final Report

Dr. Mark Richardson, Numerical Algorithms Group, Manchester
Dr. Grenville Lister, NCAS-CMS , University of Reading
Dr. Glenn Carver, ECMWF, Reading

Abstract

The widely used Integrated Forecasting System (IFS) meteorological software is now licensed for academic use as OpenIFS. However, to improve performance, additional libraries are required. In this report the installation and verification of support libraries (GRIB_API and FDB) on the HECToR Cray XE6 are described. The findings are presented for investigations into the LUSTRE file striping, OS page sizes, the number of MPI tasks used for writing, and configuration parameters for the FDB library.

Contact: g.m.s.lister@reading.ac.uk, glenn.carver@ecmwf.int

1 Introduction

The Integrated Forecasting System (IFS) is a well-known highly developed software suite owned by European Centre for Medium-Range Weather Forecasts (ECMWF - www.ecmwf.int), and used by them and other meteorological offices for medium-range weather forecasting. OpenIFS is a version of the software that has been recently made available to academic researchers. It will provide new and significant scientific opportunities for research into: weather and weather related processes, short to seasonal range forecasting, and forecast verification The IFS model uses a parallel I/O system operationally at ECMWF. Without this system the academic version, OpenIFS, will pass all I/O through the master task - limiting the types of experiments that can be carried out on HECToR. HECToR is a Cray XE6 with 90112 cores where computation is done on dedicated nodes connected by the Gemini interconnect.

This project will address this I/O deficiency by implementing the necessary parts of the ECMWF I/O software on HECToR to provide: (i) a parallel I/O capability for the OpenIFS model (FDB) (ii) a user application for managing and accessing the model output (MARS). This project prepares the way for the use of OpenIFS by installing support libraries and recording its performance with them.

Recommendations are made for the operation of the OpenIFS software on Cray XE6. Tuning optimizations that have been investigated include: the Lustre stripe size and count, the hugepage setting and the FDB configuration. This report details that work and documents four work packages making up a HECToR dCSE project. They are: the installation of the support libraries: GRIB_API and FDB (reported in section 2); confirmation that the OpenIFS software produces the same result with FDB as without FBD (section 3); an assessment of the performance of simulations using the default settings (section 4); and modification of certain parameters to see if the performance of the simulation can be improved (section 5.) Finally, we give our recommendations and conclusions.

2 Build and install support libraries

2.1 Prerequisite software

This project uses a version of FDB that is the latest development version that will soon be in operational use. The FDB library depends on the GRIB_API and ECKIT library. These must be built first. The OpenIFS software also links against the GRIB_API as it can be selected for use independently of FDB.

Package accounts exist on HECToR to manage the installation of any software needed for wider use than by one person. For the OpenIFS software package, the home directory is /home/y07/y07/oifs and the work directory is /work/y07/y07/oifs. A directory has been created under /work/.y07/y07/oifs results of the builds are installed. Hence the “prefix” for any configuration of a build includes the “install” directory.

Two implementations of the library have been built. These should be built and installed separately each time there is an update or new version released. The intention is to use the HECToR default environment to build the library, although during the project the defaults changed and the most recent (October 2013) environment is for CCE 8.1.8.

A separate build of the software was done using the GNU compiler system to allow GRIB_API tools to be used in the serial queues for such things as post-processing and job preparation. Note that the target architecture for those libraries is the “Istanbul” hex-core of the login nodes (and serial queues).

2.2 Build procedure

Occasionally the GRIB_API, FDB and ECKIT software will need to be updated. This is usually at the behest of ECMWF. The process follows a typical software build. First update the source code, then generate the makefiles. The makefiles for GRIB_API are created with GNU Autotools, the configuration step is detailed in the appendix A. For ECKIT and FDB the makefiles are generated with CMake. Some of the configuration steps do small compilation tests to determine the build environment and set values for compilation flags. This would cause the build to fail as the process is a cross compilation and some programs will not run on the login nodes. A tool chain file has been created for FDB, which by-passes those tests and returns a "success" result and the appropriate settings. During the configuration the destination for the installations is nominated as a place the compute nodes will be able to access. This is for the deferred tests that will be built on the compute nodes.

After the successful build and installation, three module files were created so that the CCE version of the libraries can be used more easily when building OpenIFS.

module load grib_api_cce	to specifically set the GRIB_API environment
module load fdb_cce	to specify the FDB environment
module load openifs_cce	to setup the OpenIFS, invoking grib_api and fdb

The module files have been placed in the modules sub-directory of the "oifs" account. A user will access them with "module use ~oifs/modules". The numbering was started at 0.0.1 for the first version and subsequent developments are indicated by incrementing the third "revision" digit. The working version was established with 0.0.2, which associates the correct version of grib_api_cce and fdb_cce with openifs_cce. More recent developments have been encapsulated in version 0.0.3. It is good practice to interrogate the modules to determine the prevailing default ("module avail openifs_cce").

Testing the libraries built for the "front-end" (login nodes) is straightforward apart from the change of compiler to GCC. The compute nodes are different to the login nodes so tests have to be launched with the job scheduler. To use the libraries on the compute nodes the modules have to be loaded within the job script as well.

2.3 MetView visualisation software

It became clear that a method for quickly looking at the results would be required by users. This is usually done using MetView [ref 1]. The software was downloaded and a build was attempted. This has proved difficult due to many dependencies not being available on the target system. The work is extra to the DCSE so a separate query has been set up with the HECToR help desk. Meanwhile a batch version (no GUI) was successfully built for the serial queue and is available in the ~oifs directory:

/usr/local/packages/oifs/metview/4.3.9/gcc_4.7.2-mc6/bin

There was very good support from the developers at ECMWF to get to this stage.

3 Confirm that OpenIFS works with FDB

To confirm that the installations have been successful and that OpenIFS works with FDB, three test cases (T159, T511, and T1279) have been provided. It is possible to run each case either without FDB activated (LFDBOP=F[alse], the default) or with FDB activated (LFDBOP=T[rue]). The mode is changed through an entry in a namelist file.

The default mode of operation is for OpenIFS to write GRIB files each output time step. There are two GRIB files produced on each specified time step; one contains grid point data and the other contains spectral data, the files are named accordingly. For example:

ICMGGfwj8+000024 for grid point data

ICMSHfwj8+000036 for spectral data

A disadvantages of this output convention is the fact that many files are created for any one simulation. Also the files are written using a master writer which requires a collection step for data from the other MPI tasks. The advantage is that the files are portable and conform to a widely used standard. More detail is given in Appendix B.

When the FDB option is activated while still in the GRIB format, the detailed structure of the files is different and thus the directory structure is changed also. Instead of generating "per time step" GRIB files in the case directory, a subdirectory is named "fdb<RUNID>" and the FDB system creates a further sub-directory (for example " rd:oper:fwj8:20121027") with a name related to the simulation start date and experiment type as seen in table 1.

Table 1: typical files created by the FDB process for case T511 in the fdb sub-sub-directory	
Size in bytes	File name
165953536	0000:fc:sfc.0.20131016.084031.7773890805760.dat
65536	0000:fc:sfc.0.20131016.084031.7773890805761.idx
267	0000:fc:sfc.0.20131016.084031.7773890805761.idx.files
165953536	0000:fc:sfc.0.20131019.164347.88669599825920.dat

Thus the comparison between non-FDB and FDB output is not straight forward. However, in this section the baseline cases are presented and a method for comparing was carried out and gave identical information. This is expected as the content of the FDB files is GRIB encoded data simply in a different structure to the "standard" OpenIFS output.

3.1 Baseline of test cases

The source code for OpenIFS was extracted from the University of Reading repository (PUMA), built using the new modules and support libraries, and basic testing was done. Details of building IFS are given in appendix B. Later investigations into tuning the performance were done with a revised T1279 referred to as T1279_op. The result of the build is a binary (master.exe) that can be launched from within a "standardized" script named **ifs_run.sh**.

The early version of OpenIFS can only generate files in GRIB format in the form of a spectral file and a grid-point file. These are written separately for each output time step as specified by a user. The development version used for this project includes the ability to write FDB files.

For example, the T159 case, as it was supplied, generated 15 files, 2 each for 7 writing steps and a log file. The simulation wrote results data every 72 model time steps. A coarse level comparison is obtained by summing up the sizes of files in the results directory and comparing to the data volume of the FDB archive. In each case the total number of bytes matched. We ensured the results were identical at the bit level by concatenating the FDB GRIB files using the GRIB_API tools to extract individual fields and again using GRIB utilities to compare these fields with those extracted from the non-FDB GRIB output from the otherwise identical run. [CAUTION: use the GRIB tools built for login nodes to inspect the results files]. In all cases considered we obtained exact matches.

Table 2 shows the baseline values for a non-FDB run of the three test cases (T1279_op was not run without FDB) taken from the standard log file of the run. It is an average of the days completed over the wall time.

Table 2: comparing sizes of results files between non-FDB and with-FDB for each test case.				
Case	**T159**	**T511**	**T1279**	**T1279_op**
MPIxOMP	**16x8**	**128x8**	**512x8**	**512x8**
Total cores	**128**	**1024**	**4096**	**4096**
Non-FDB				
Non-FDB file size per time step	14MB+31MB (45MB)	133MB+248MB (481MB)	800MB+1.6GB (2.4GB)	-
Directory size at end of run	315MB	2.6GB	16GB	-
Forecast days per day non-FDB	3372	679	284	-
With-FDB				
Directory size at end of run	315MB	2.6GB	16GB	324GB
Baseline wall time Minutes	5	9	17	26
Forecast days per day with-FDB	3347	691	292	193

The fourth case, T1279_op, was provided later in the project and as such required a change to the OpenIFS software. This meant that the name-list had to be changed for the earlier cases to make them compatible with the new version of OpenIFS. The test cases had to be modified in the following way: the name-list variable NFPPHY was reduced from 88 to 86 because the final two entries for the variable MFPPHY were unknown to the updated software.

3.2 Scaling exercise

The early test case T1279 was used for a scaling investigation.

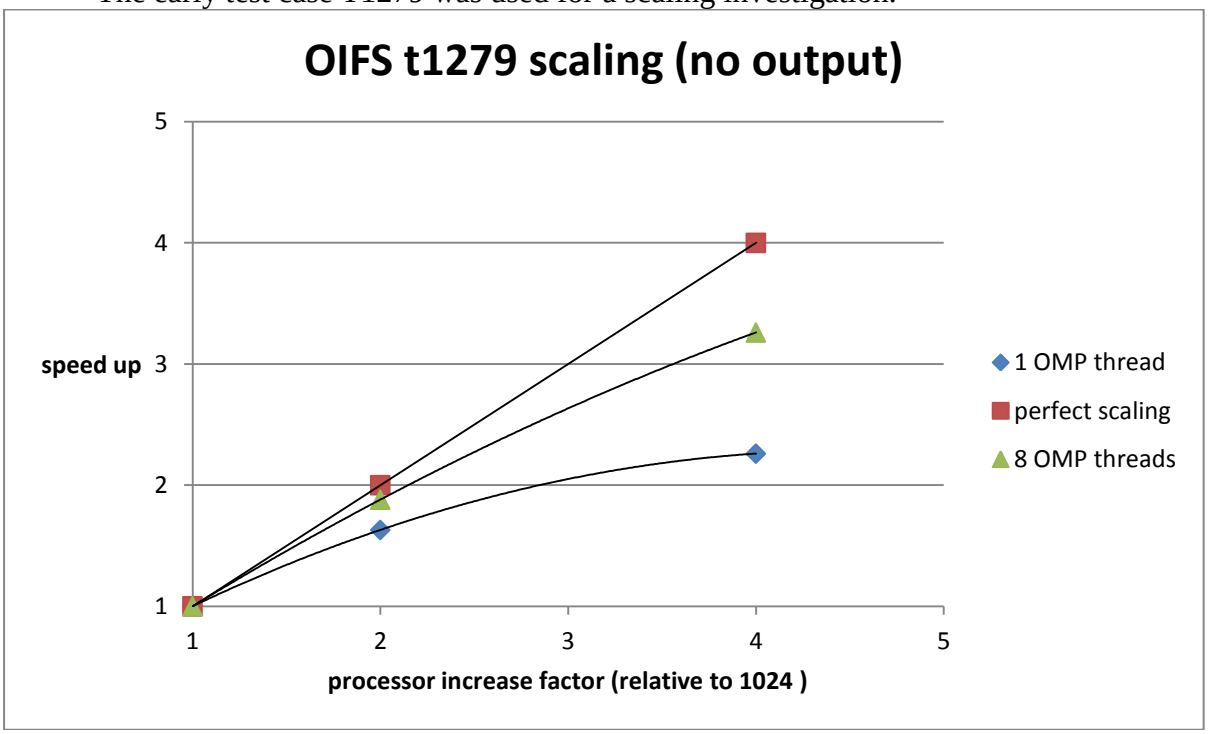


Figure 1 shows that the pure MPI scaling is moderate up to 4096 MPI tasks. Accelerating this with OpenMP threads approaches the idealized "perfect scaling" line. Some of this effect is due to fewer MPI tasks being used and thus less communication traffic between nodes.

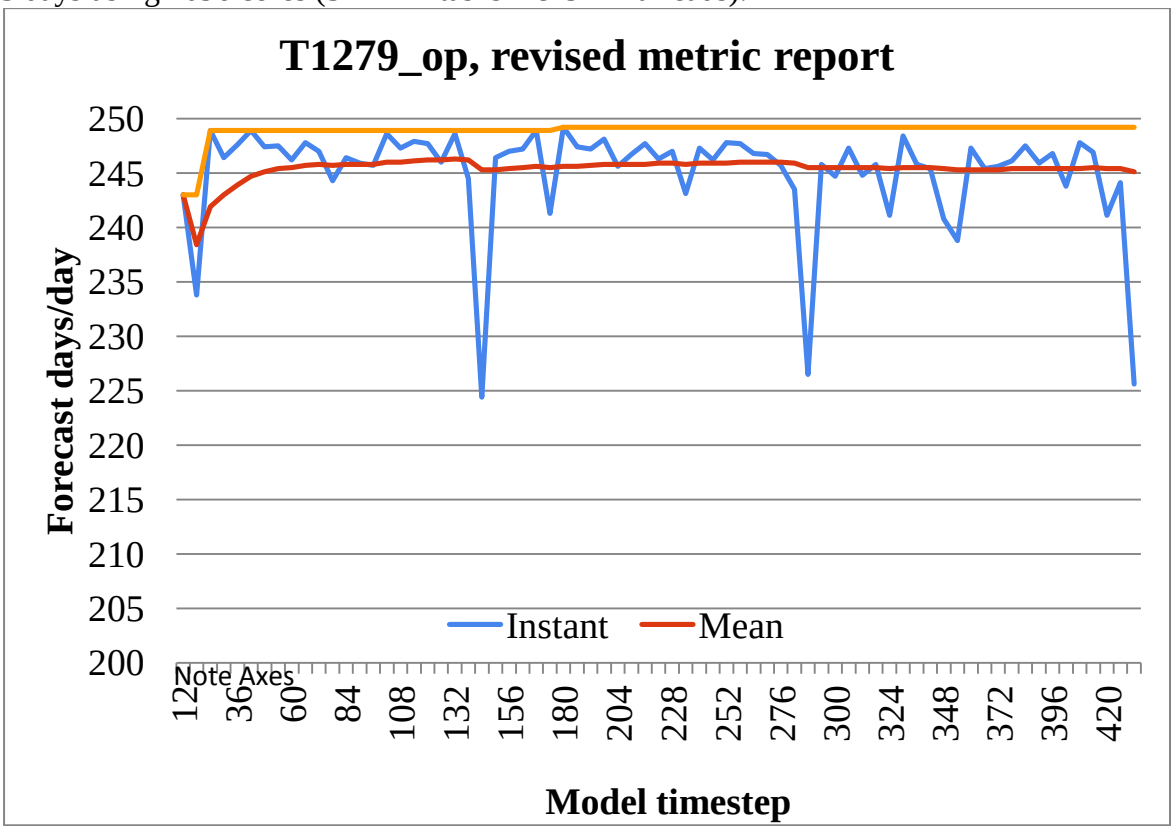
4 Establish IO metrics

Three of the supplied cases were purposely simple to help on the occasion where a quick turnaround was significant. Generally it was possible to run within the 20 minute queue and thus the wait time was not too challenging (the worst seen was 12 hours due to the busyness of the machine). Cray performance analysis tools were used to inspect the write rates for the files in both cases; without FDB and with FDB. The without FDB generates two files per writing step (not all steps write data). As described in section 3, the with-FDB mode generates 3 files per writer. Only the "dat" files are added to at each write point. The number of writers can be varied as discussed in section 5.3 of this report.

4.1 Built-in instrumentation

The metric that has most significant indication of performance is the "forecast days per wall clock day". That value is influenced by several factors: the size of the case, the complexity of the case and the number of cores working on the problem. The number of cores is determined by the settings NPROC (set in the namelist file) and NTHREAD is set as a job entry which is assigned to the OMP_NUM_THREADS environment variable just before the run starts.

Figure 2: plot of the per step metric forecast days per day for T1279_op simulation of 3 days using 4096 cores (512MPI tasks x 8 OMP threads).



Late in the project developers at ECMWF inserted extra instrumentation in the form of writing a mean, peak and instantaneous forecast days per day on each model time step.

Figure 2 the regular dip in instantaneous forecast days per day due to a regular additional calculation during that model time step (every 24 hours).

4.2 Cray performance analysis tools

Several experiments were done using the Cray PAT toolkit. Initial sampling and tracing provides some insight into the scaling of the code. Figure 3 and figure 4 show the profile for the user section of code for the two smaller cases. They are noticeably similar and imply that the scaling is quite good. It was noted by the ECMWF developers that the master loop 17 should not really feature so high as it is trivial. However, it is indicative of the short nature of the runtime of these cases and that initialization will dominate.

Figure 3: T159 tracing report (partial) from Cray PAT

```
T159 NPROC=16 NTHREAD=8
|  65.6% |  90.686457 |       -- |     -- | 670210.4 |USER
||-------------------------------------------------------------------------
||  13.7% |  18.901611 | 1.523439 |   8.0% |      1.0 |master_.LOOP@1i.27
||   7.6% |  10.466567 | 0.334678 |   3.3% |     73.0 |ec_phys_drv_.LOOP@1i.409
||   6.1% |   8.475113 | 0.720776 |   8.4% |    149.8 |radlswr_
||   5.0% |   6.881496 | 0.287358 |   4.3% |    876.0 |vdfmain_
||   4.2% |   5.867587 | 0.116884 |   2.1% |    876.0 |cloudsc_
||   3.9% |   5.406474 | 0.113219 |   2.2% |     73.0 |cpg_.LOOP@1i.666
```

Figure 4: T511 tracing report (partial) from Cray PAT

```
T511 NPROC=128 NTHREAD=8
|  64.6% | 317.214155 |        -- |     -- | 2747397.5 |USER
||-------------------------------------------------------------------------------------
||   9.1% |  44.793452 |  1.897469 |   4.1% |     289.0 |ec_phys_drv_.LOOP@1i.409
||   7.4% |  36.505546 |  1.087992 |   2.9% |       1.0 |master_.LOOP@1i.27
||   6.7% |  32.714537 |  3.603615 |  10.0% |    4046.0 |vdfmain_
||   5.7% |  27.784058 |  0.866872 |   3.0% |    4046.0 |cloudsc_
||   5.4% |  26.269358 |  0.949899 |   3.5% |     289.0 |cpg_.LOOP@1i.666
```

The file access pattern can also be monitored with Cray PAT. The sequence in figure 5 is a recommended process for extracting this information; in particular step 4 is where the choice of experiment is influenced.

Figure 5: The *perftools* module MUST be loaded before the build of OpenIFS commences.

module load perftools

(1) Instrument the executable
pat_build –O apa master.exe (ISSUE)

Normal sample - for where time is spent in this short run
Apa file ascii editable human readable – options for pat_build

(2) APA guides a subsequent trace experiment
pat_build –O (apa file)
(3) Followed by IO expt using master.exe+apa
(4) Edit the apa file to add io groups
-g sysio,stdio,ffio,aio,
-g mpi
-g omp

4.2.1 PAT report for OpenIFS without FDB

After a successful run of the test case for the tracing experiment, pat_report is used to generate a text file. The text file is quite detailed and incorporates several sections for

different performance measurements. An example of the section on files, writing information to disk, is shown in the following excerpt.

In table 3 the example shows the two different file categories “GG” and “SH” for T159. The write rates vary greatly from a few MB/s to 100s of MB/s with no clear trend for if the file was written early of late in the simulation. The write rates are different even for files with the same number of bytes.

Table 3: File Output Statistics by Filename T159 non-FDB

Write time seconds	Write MBytes	Write Rate MBytes/sec	Writes	Write Bytes per call	File Name
27.281	30.28	1.110	77.0	412462.55	./ICMSHfwj8+000048
5.452	30.28	5.555	77.0	412462.55	./ICMSHfwj8+000000
3.790	13.82	3.647	41.0	353638.93	ICMGGfwj8+000024
2.904	30.28	10.427	77.0	412462.55	./ICMSHfwj8+000036
2.753	30.28	10.998	77.0	412462.55	./ICMSHfwj8+000024
2.071	30.28	14.620	77.0	412462.55	./ICMSHfwj8+000060
0.110	13.82	124.739	41.0	353638.93	ICMGGfwj8+000036
0.100	13.82	137.674	41.0	353638.93	ICMGGfwj8+000072
0.091	30.28	330.962	77.0	412462.55	./ICMSHfwj8+000012
0.068	11.17	162.368	41.0	285687.51	ICMGGfwj8+000000
0.062	30.28	484.929	77.0	412462.55	./ICMSHfwj8+000072
0.048	13.82	286.339	41.0	353638.93	ICMGGfwj8+000060
0.047	13.82	289.694	41.0	353638.93	ICMGGfwj8+000048
0.041	13.82	336.466	41.0	353638.93	ICMGGfwj8+000012

4.2.2 Cray PAT statistics for case T159 FDB output

A similar tracing experiment was performed where FDB output was enabled. This also showed large variation between the write rates for the files as shown in table 4. The main difference between the non-FDB and with-FDB run is that with-FDB enabled only one file per writer task is written but it is incremented on the subsequent writes. The run was performed with the default Lustre stripe count=1 and stripe size =4.

Table 4 : file access (writes) statistics for T159 with FDB activated, reported by CrayPAT

Write time seconds	Write MBytes	Write Rate MBytes/ sec	Writes	Write B/Call	File Name
0.103	19.89	191.99	812.0	25695.84	/:fc:sfc.8.20131018.112231.125937031053312.dat
0.099	18.06	181.97	756.0	25058.20	/:fc:sfc.14.20131018.112231.123394410414080.dat
0.091	20.76	220.45	826.0	26366.14	/:fc:sfc.4.20131018.112231.125464584650752.dat
0.070	20.87	295.07	840.0	26063.24	/:fc:sfc.3.20131018.112231.127242701111296.dat
0.067	19.89	296.78	812.0	25695.84	/:fc:sfc.7.20131018.112231.125477469552640.dat
0.060	20.99	347.69	840.0	26204.65	/:fc:sfc.1.20131018.112231.127234111176704.dat
0.054	19.26	353.77	784.0	25761.96	/:fc:sfc.10.20131018.112231.125945620987904.dat
0.053	19.07	357.10	798.0	25068.72	/:fc:sfc.9.20131018.112231.125941326020608.dat
0.052	18.13	347.52	756.0	25150.31	/:fc:sfc.13.20131018.112231.123390115446784.dat

Files in the FDB directory infrastructure include the metadata in the filename which makes it difficult to visually scan a directory quickly, also repeat runs generate a fresh set of

"dat" files. Using a large number of MPI tasks will default to writing a large number of dat files.

5 Investigation of variation from the default values

Several parameters can be changed to investigate the performance of OpenIFS. These include the LUSTRE tuning parameters; the size of virtual memory pages and the FDB writing mode. This section describes the effect of changing these parameters. In the description of building the OpenIFS (Appendix B) there is a reference to page size; this section includes a description of how page size can be changed and what effect it has on the simulation. The baseline for the work is established with the three test cases using a default value of page size of 4096 bytes and the default LUSTRE stripe count of 4 and stripe size of 1MB.

5.1 LUSTRE

Lustre® is a high-performance, massively-scalable, POSIX-compliant network file system designed for high-performance compute clusters (see http://www.Lustre.org).

Lustre file systems are made up of multiple services typically distributed across multiple nodes. A file system is comprised of; a Meta-Data Target (MDT), which stores directory and file meta-information such as file ownership, timestamps, access permissions, etc. and a series of Object Storage Targets (OST), which hold the file data in one or more objects.

On the HECToR system there are two LUSTRE file system partitions; a general user partition where the default stripe count is 1 and a stripe size of 1MB. The second partition is for NERC researchers where the default is a stripe count of 4 and stripe size of 1MB. This project work was served by the NERC partition due to the funding nature of the research. There are 84 OSTs.

There are several ways to configure LUSTRE including (i) as an administrator setting an optimum for the installed hardware (ii) as a user choosing the striping pattern (iii) as a user adding code to set striping patterns from within a simulation. The majority of this work was done with (ii) as the system hardware has been set up during installation, the API work was deferred until it was clear if any gains can be achieved with (ii).

There are several administration commands not for general use (**mkfs.lustre**, **tunefs.lustre, mount.lustre, lctl**). However, the user-level interface to control Lustre-specific information for individual files is **lfs**. This command is used with several options (sub-commands) to control the stripe count and stripe size for directories. It can be used to create an empty file with a preset lustre configuration but it cannot be used to modify existing file striping. When a directory has been configured the content inherits the same striping, so new file creation from within the program will acquire the same stripe attributes.

lfs setstripe –c 8 –s 16M <directory name>

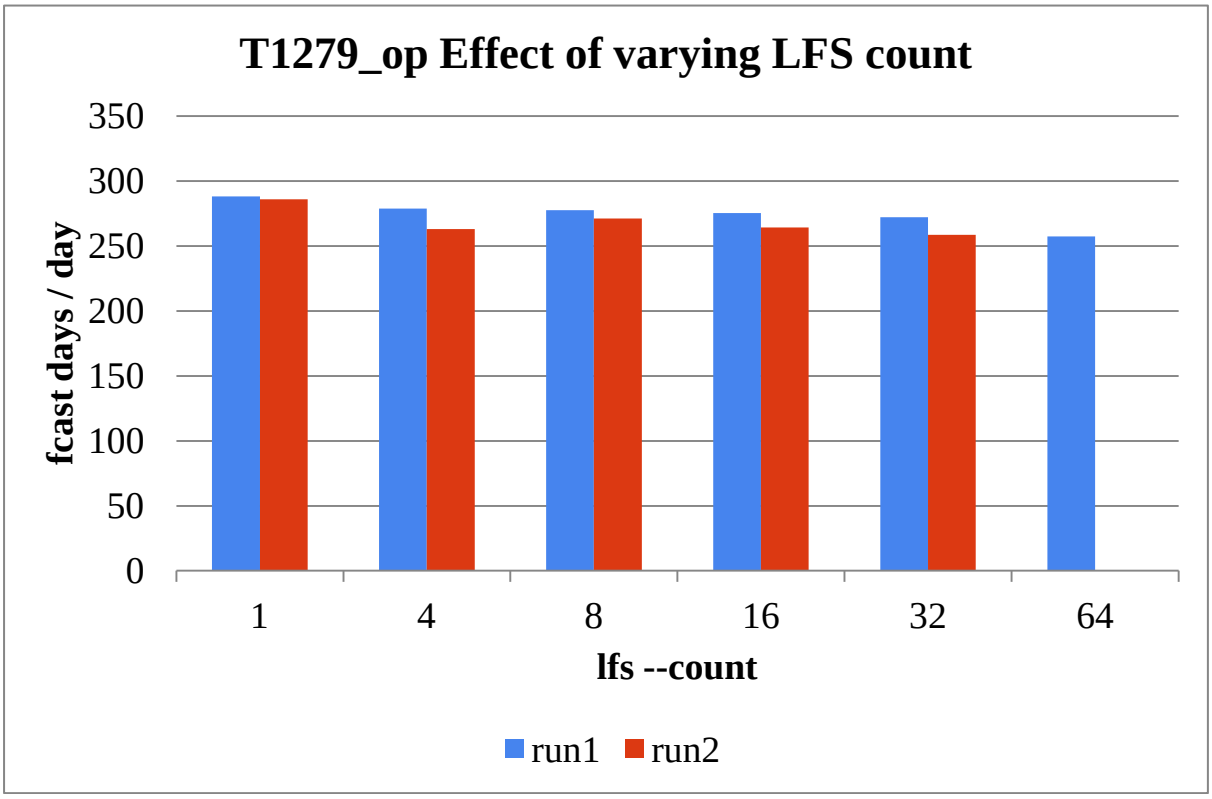
will set the stripe count to 8 and the stripe size to 16M on a directory.

The work for this section was done using the T1279_op and along with the testing of varying hugepage size (next section). Figure 6 and figure 7 are representative graphs of how the run times were affected by changing the Lustre parameters for a specific huge page value of (2MB).

5.1.1 Changing the number of stripes

The default setting is a stripe count of 4. The setstripe command is applied to the case directory just before the run so that any freshly created files will adopt the revised attribute.

Figure 6 A series of experiments investigating lustre stripe count for fixed huge page of 2M and stripe size of 4MB.



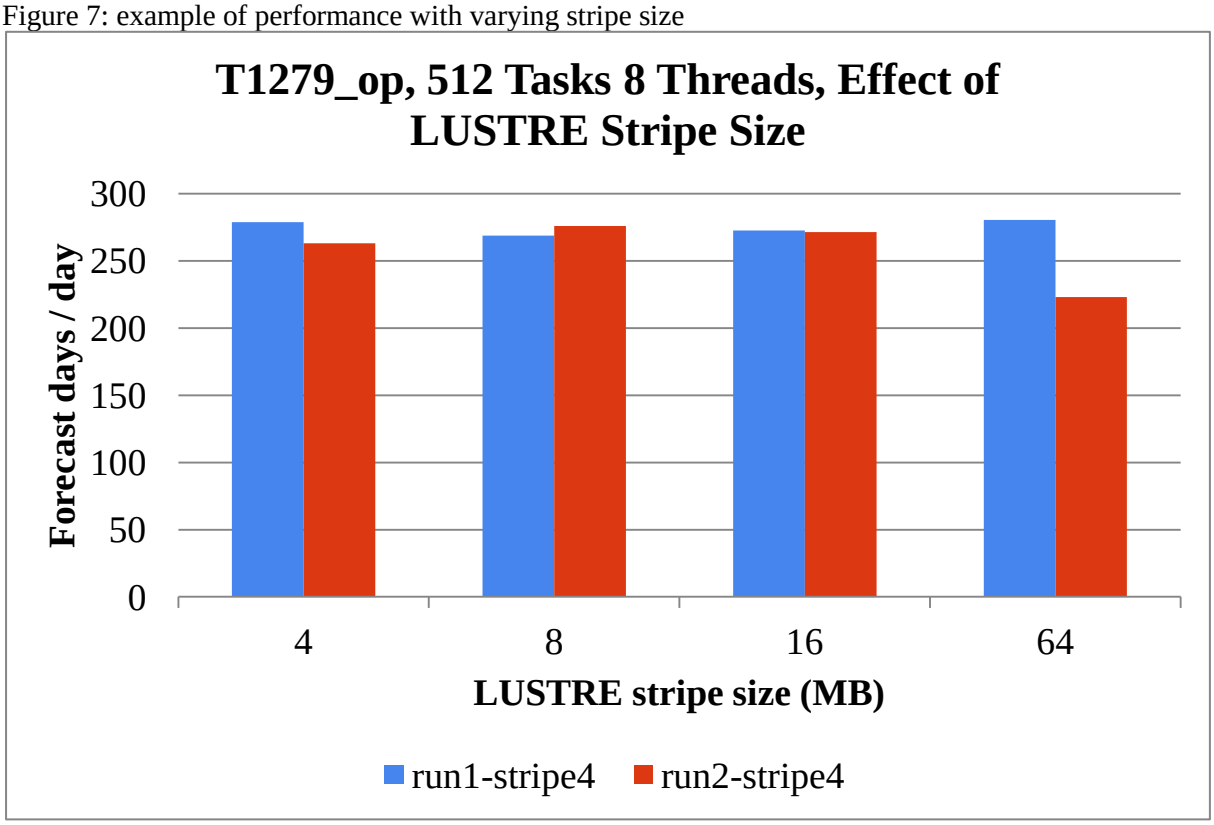
The default setting is not ideal for this application. The performance improves when the setting is for more stripes and when reverting to a single stripe. Also this chart shows the variable nature as the repeat runs show a consistent slower performance.

5.1.2 Changing the size of the stripe

The default setting is a stripe size of 1MB. The **setstripe** sub-command is used to change the attribute on the case directory before the run. There is some variation between the runs that is inconsistent, with second run occasionally being faster than the first. For one set of runs the 8MB stripe is the minimum performance whereas for the second runs the 8MB is the peak of performance.

This has shown that a small change in performance can be achieved when following the general advice for the case where each MPI task has a dedicated file i.e. set the stripe count to unity. The stripe size should not make any difference when operating such a configuration. One difficulty of doing this is down to the fact that the default stripe count for ("n02") projects is 4; users should be advised to set the stripe size on the case directory to unity.

Figure 7: example of performance with varying stripe size



5.2 Hugepages

Huge pages are virtual memory pages which are larger than the default base page size of 4Kbytes. Huge pages can improve memory performance for common access patterns on large data sets. They can also increase the maximum size of data and text in a program accessible by the high speed network. Access to huge pages is provided through a virtual file system called *hugetlbfs*. The executable has to be linked to the library to use huge pages.

There are six additional page sizes that are classed as “huge pages” that can be accessed through module files. The module file sets the necessary link options and run time environment variables for the specific huge pagesize choice. The six modules are craype-hugepages128K, craype-hugepages512K, craype-hugepages2M, craype-hugepages8M, craype-hugepages16M, craype-hugepages64M.

When an executable has been built with an activated huge page module then it is possible to vary the pagesize at runtime. The recommendation is to compile OpenIFS with the 2M huge page active and thus be able to vary the page size. i.e.

module swap PrgEnv-pgi PrgEnv-cray
module load craype-hugepages2M

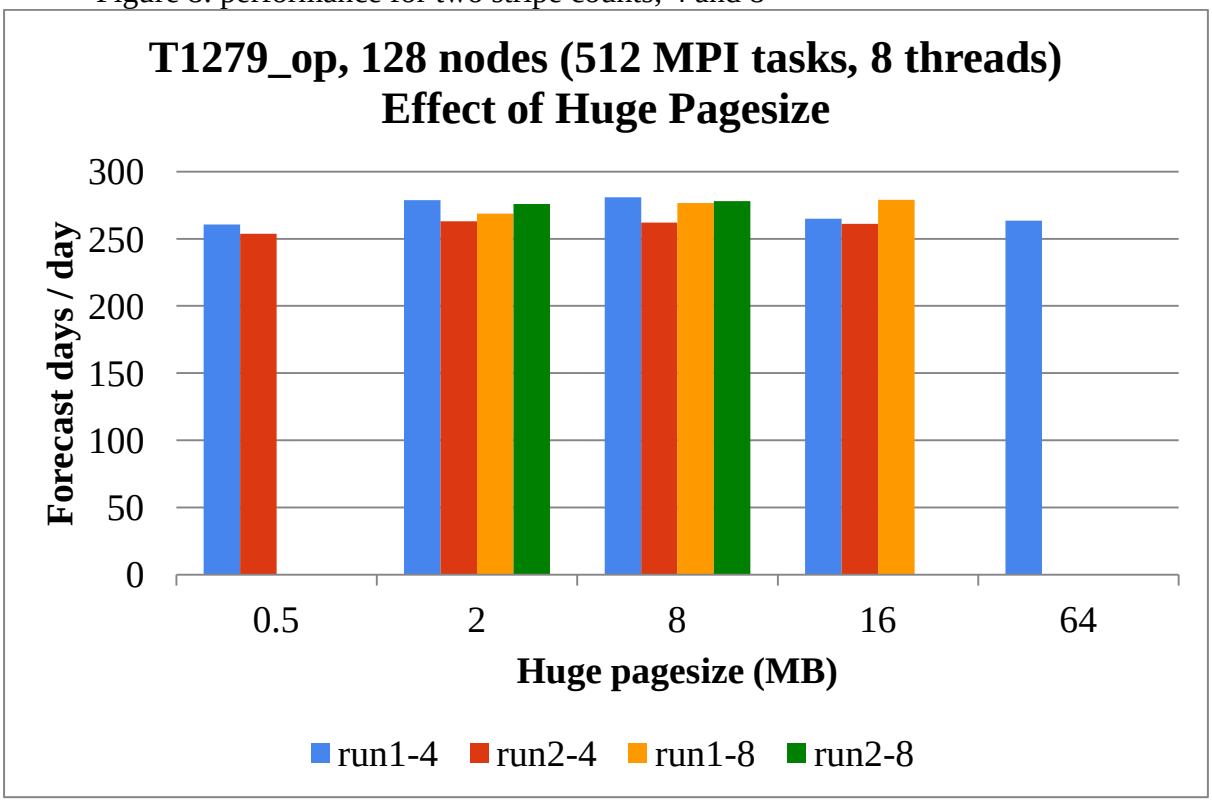
Then build as usual.

Of the environment variables that are set by the module, two are slightly different depending on the chosen page size. These are shown in table 5.

Table 5: environment variables for huge page compilation and running	
Value for HUGETLB_DEFAULT_PAGE_SIZE	Variable name e.g. HUGETLB16M_POST_LINK_OPTS (i.e. -Wl,--whole-archive,-lhugetlbfs,--no-whole-archive -Wl,-Ttext-segment=0x20000000,-zmax-page-size=0x20000000)
128K	HUGETLB128K_POST_LINK_OPTS
512K	HUGETLB512K_POST_LINK_OPTS
2M	HUGETLB2M_POST_LINK_OPTS
8M	HUGETLB8M_POST_LINK_OPTS
16M	HUGETLB16M_POST_LINK_OPTS
32M	HUGETLB32M_POST_LINK_OPTS
64M	HUGETLBM_POST_LINK_OPTS

The various page sizes were tested with two lustre striping strategies; the default stripe count of 4 and a stripe count of 8. The stripe size was left at the default of 4MB. Figure 8 illustrates the difficulty in repeatability as only "run1-8" shows a distinct (monotonic) increase in forecast days per day with increasing page size.

Figure 8: performance for two stripe counts; 4 and 8



As a consequence it is recommended that the "huge page size" should be set to 2MB as no more significant gain is indicated for larger page sizes. Other combinations were investigated (shown in table 6) and the chart in figure 8 is representative of the other results.

Table 6: all stripe count =4 and the number is recommended sequence for investigation					
Lustre stripe size	craype-hugepages512K	craype-hugepages2M	craype-hugepages8M	craype-hugepages16M	craype-hugepages64M
1MB					
2MB		6			
4MB	2	1	3	4	5
8MB		7			
16MB		8			
64MB		9			

5.3 Varying the Number of Writers

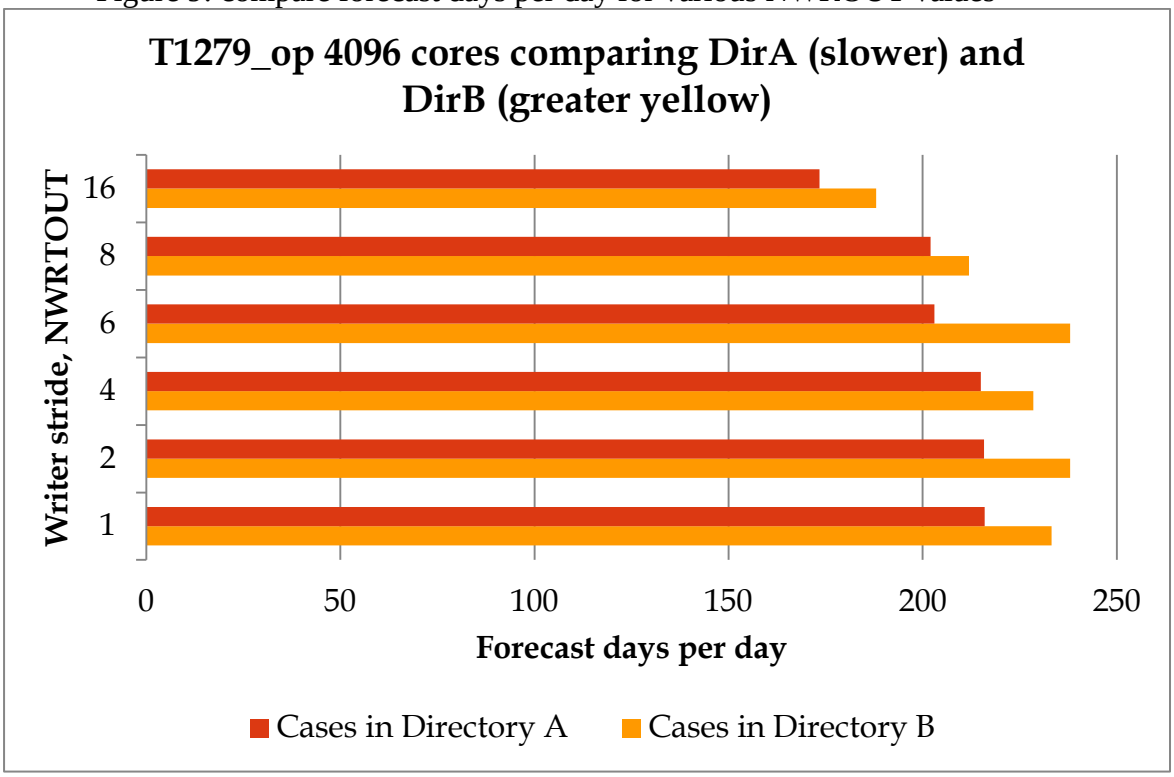
The number of MPI tasks that write FDB data can be changed to be a sub-set of the enrolled tasks. The tasks are selected by striding through the task IDs at an interval set in the name list file (fort.4). The variable NWRTOUT is the interval of MPI tasks between writers. This is by default unity where all MPI tasks are involved in writing FDB data files. If the value is changed then some of the MPI tasks that are on the same node can be used to help encode data into the GRIB format.

The encoding can be shared by the MPI tasks on a common node that are not being used to write data. For example, one might set the writer stride to match the number of MPI tasks per node so that only one writer sits on a node. This limits the number of streams of data being written to disk from the node. The remaining MPI tasks can be used to "help" the writer encode the data for GRIB format. However, data has to be communicated in several ways; one for the encoding and another for accumulating the subset of data onto the writer MPI task.

Table 7: Case T1279_op repeated runs with FDB, decreasing number of writers									
Writer stride	Number of data files	Typical Size per File (GB)	Total size files in directory (GB)	Case results written to directory A			Case results written to directory B		
				Wall time minutes	F/D peak	F/D (average)	Wall time minutes	F/D peak	F/D (average)
1	512	1.9 +0.5	218	16.0	216	198	14.3	233	221
2	256	2.2 +1.1	218	15.8	216	203	14.5	238	220
4	128	2.0	218	16.5	215	192	14.5	229	219
6	86	3.8	218	17.6	203	193	14.5	238	220
8	64	4.5	218	16.5	202	191	15.8	212	202
16	32	7.5 +7.1	218	20.1	173	153	19.5	188	160
+there were some files with significant size difference									

The values from table 7 are presented in graphical form in figure 9. There is a significant bias to directory B which is difficult to explain without much more extensive testing.

Figure 9: compare forecast days per day for various NWROUT values



The general outcome is that decreasing the number of writers has little beneficial effect on the simulation time; in fact the runtime is increased for the majority of cases. An unexpected behaviour is that there was a distinct difference when the simulation results were written to a different directory. This was noticed when the identical case was run simultaneously with different strides in two separate directories. This was a consistent result when those run configurations were switched to the alternate directory.

For a stride greater than four the writer will be on a different node as these runs have been configured for 4 MPI tasks per node and OpenMP thread count of 8. The stride of 6 ws tested because it results in 85.6 writers (actually 85) which is close to the number of OSTs (84). The results for directory A show that indeed a change is noted between 4 and 6 however this is more likely attributable to there being some nodes without writers, i.e. more data (work) for the shared cores on one node to encode.

5.4 Tunable FDB parameters

When the FDB library is installed it is possible to set the runtime parameters in

$FDB_HOME/etc/config/fdb/fdbRules

The file includes four settings that can affect the behaviour of FDB, shown in table 8. The fdbAsyncWrite parameter is set to “false” during the installation by default and implies that the testing for section 5.1 and section 5.2 were done with no asynchronous file handling. The configuration file is in a protected location (i.e. only the installer may change it), so any changes will affect all running simulations (i.e. other users). This work will be useful to establish which setting is appropriate for the Cray XE6. The configuration file was changed to set the parameter to TRUE. However, that resulted in the simulations crashing.

Table 8: configuration parameters in the FDB rules file			
Parameter	Value on HECToR	Options	remarks
fdbAsyncWrite	**False**		Activates the async writing.
		fdbNbAsyncBuffers	4, number of buffers for async io
		fdbSizeAsyncBuffer	(64*1024*1024), size in bytes of each buffer for async io
doubleBuffer	**0**		Use cyclic buffers for data transfer.
		bufferSize	67108864 (64*1024*1024)
		doubleBufferSize	524288 (0.5*1024*1024)
		doubleBufferCount	20
blockSize	**-1**		Add padding to index access to match file system block size given
syncDirOnFileFlush	**True**		force a flush of directory on flushing files

6 Summary

The support libraries have been built and installed on the Cray XE6. They have been built to support two compilations; one using Cray compiler environment (CCE) and the second using GNU complier suite (GCC). The OpenIFS software has been built and coupled to the FDB library. It has been shown to give identical results to those of the "traditional" GRIB serial writer files. This was expected because it uses the same GRIB encoding of the same data.

The metric for four cases has been established as the forecast days per day and this has been used to record the performance of the FDB-enabled software. The timing and tuning results are indicative of a busy multiuser machine.

The differences achieved by changing the LUSTRE parameters are not very noticeable compared to the variation seen in multiple runs. The runtime appeared to be affected also by the case directory. The lack of significant speed up from LUSTRE parameters is attributable to the fact that although FDB is identified as a parallel IO method the parallel nature is between the data streams rather than an equivalent MPI-IO call to parallel file writing. If the number of IO streams is similar to the number of OSTs then it is unlikely that there would be any performance gain. In that case the striping is more likely increase the load on the OSTs and cause contention.

The recommendation is to use one writer per node although it is expected that there is an increase in intra node communications for the GRIB encoding.

Hugepages were investigated and a subtle improvement discerned for the situation using 2MB pages. The advice for configuring OpenIFS is to use the hugepages2M module and then set a directory stripe pattern with one strip of size 4MB.

ECMWF hae found this work beneficial as it is the first port of these libraries to non-IBM systems. It has also extended their knowledge to use heterogeneous CMake (cross-compilation). The development machines have used GFS so the results of the behavior of Lustre has been very helpful. The "reduced writer" section of code had been dormant but this work has re-activated.

7 REFERENCES

[1] Metview 4 Wiki https://software.ecmwf.int/wiki/display/METV/Metview

Appendices

Appendix A: Build and installation of the support libraries
Appendix B: Result of tests of installations
Appendix C: Using OpenIFS on HECToR

Appendix A. Build and install support libraries

OpenIFS is a weather forecasting system that has established GRIB as its file format. The GRIB files are writing at time intervals specified by the user. Alternative results storage is available in the form of "FDB" (Fields Data Base). For the normal operation GRIB files are written at specified intervals and a naming convention identifies what information is in the file (a form of metadata). The OpenIFS software has to be linked with the GRIB_API library to write GRIB files. To use the more efficient parallel output method, FDB, the FDB library has to be available; it also uses the GRIB format and will also need to be linked to the GRIB_API library. Both of these support libraries have been installed and this section describes the work done to achieve that. A more detailed description of the configuration of the libraries is given in the appendix A.

7.1 OIFS package account for software management

An account has been set up as a package account. Package accounts exist on HECToR to manage the installation of software for wider use than one person. For OpenIFS software package the home directory is /home/y07/y07/oifs and the work directory is /work/y07/y07/oifs. A directory has been created in the work directory for the results of the builds to be installed. Hence the "prefix" for any configuration of a build includes the "install" directory.

7.2 The build procedure

Ensure the software is up to date, set up the build environment (CCE) and then build GRIB_API, this uses the GNU Autotools style configure script, followed by ECKIT, then FDB, note that these use CMake. Directions for doing this are in the ~/NOTES directory of the oifs package account. Update the notes to reflect the changes. The build time for OpenIFS is less than 20 minutes and is done on the login node hence it is a cross-compile.

Install the three packages. NOTE a similar process has been followed for the GCC build of these packages. Five modules have been created; one for each port of GRIB_API and FDB and the fifth is the OpenIFS environment that invokes the CCE versions of the GRIB_API and FDB. As newer versions or combinations of the libraries are installed the modules have to be kept up to date.

7.2.1 Update the software

Command is "git pull origin openifs" within the source code directory.

7.2.2 Configure the make files

7.2.2.1 ECBUILD

There is no actual build required for this software. It is a collection of text files, scripts and CMake files. It directs the configuration of the other software; ECKIT and FDB (and if it were to be used, GRIB_API). It has been installed below the home directory of "oifs" i.e.

/home/y07/y07/oifs/ecmwf/ecbuild.

The version of GRIB_API software is 1.9.18 supplied by ECMWF and it is built separate to the "ECMWF" software (in which there is a version of GRIB_API that can be configured and built with CMake). Other products use GRIB_API therefore it has to be built for both front-end and back-end environments.

7.2.2.2 GRIB_API

The make files for GRIB_API are created with GNU Autotools and the configuration step is:

Figure A1: Generating the makefiles f or GRIB_API (CCE compiler)

```
./configure --disable-shared --disable-jpeg --disable-python --enable-pthread \
                --prefix=/work/y07/y07/oifs/install/grib_api/1.9.18/cce_8.1.8-il
```

This configuration line is similar for the GNU compilation except that the directory representing the platform is gcc_4.7.2-mc6 to indicate the Istanbul target. The GRIB_API includes some auxiliary data that is installed alongside in: share/grib_api

The file is an include file and one has already been prepared for HECToR that can be copied into the source. It has to be a copy rather than clone (cp –p filename newfile) to trigger the make rules.

The unit testing is done on the compute node and this script is used to start that.

Figure A2: Script for unit testing on a node using as many cores as available because each test is serial

```
#!/bin/bash --login
#PBS -N GribTest_n32
#PBS -l mppwidth=32
#PBS -l mppnppn=32
#PBS -l walltime=01:00:00
#PBS -A y07

module load CMake/2.8.8
module swap cce cce/8.1.8
module li

PLATFORM=cce_8.1.8-il

# grib_api paths
export GRIB_API_PATH=${HOME}/grib_api/1.9.18/${PLATFORM}
export PATH=${GRIB_API_PATH}/bin:${PATH}
export LD_LIBRARY_PATH=${GRIB_API_PATH}/lib:${LD_LIBRARY_PATH}
export GRIB_DEFINITION_PATH=${GRIB_API_PATH}/share/grib_api/definitions

cd $PBS_O_WORKDIR
echo "Grib Unit Test"
aprun -n 1 -d32 -cc 0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30 -a xt -b make -j 16 check >
${PBS_JOBID}.out
```

The results are presented in Appendix C

7.2.2.3 ECKIT

The build of ECKIT depends on ECBUILD. After ECBUILD has been updated then the ECKIT software too can update with the same command. The make files for ECKIT are created with CMake with the following options:

Figure A3: Generating the makefiles f or ECKIT

```
cmake .. \
-DCMAKE_MODULE_PATH=/home/y07/y07/oifs/ecmwf/ecbuild/cmake \
-DCMAKE_C_COMPILER=cc \
-DCMAKE_CXX_COMPILER=CC \
-DCMAKE_BUILD_TYPE=Release \
-DENABLE_RPATHS=OFF \
-DCMAKE_INSTALL_PREFIX=/work/y07/y07/oifs/install/eckit/0.3.0/cce_8.1.8-il \
-DBUILD_SHARED_LIBS=OFF

# There is a development tool chain file
###-DCMAKE_TOOLCHAIN_FILE=/home/y07/y07/oifs/ecmwf/ToolChain-eckit.txt
```

An alternative to repeated typing of CMake options is possible with a preset text file so that fewer mistakes are made on a command line: *CmakePresetHectorEckitCce.cmake.* It should be invoked with CMake as:

cmake –C CmakePresetHectorEckitCce.cmake ../

The file content is text shown in figure A4. The form is
SET(PARAMETER value)
Note that each entry is a single line in the actual text file)

Figure A4: a file for presetting options

```
# This file is  preset for use on HECToR to build ECKIT with Cray CCE
# Mark Richardson, NAG Ltd, 2013
#
SET( CMAKE_MODULE_PATH CACHE UNINITIALIZED
/home/y07/y07/oifs/ecmwf/ecbuild/cmake )
SET( CMAKE_C_COMPILER CACHE FILEPATH /opt/cray/xt-asyncpe/5.17/bin/cc )
SET( CMAKE_CXX_COMPILER CACHE FILE /opt/cray/xt-asyncpe/5.17/bin/CC )
SET( CMAKE_BUILD_TYPE CACHE STRING Release )
SET( ENABLE_RPATHS CACHE BOOL OFF )
SET( CMAKE_INSTALL_PREFIX CACHE PATH
/work/y07/y07/oifs/install/eckit/0.3.0/cce_8.1.8-il )
SET( BUILD_SHARED_LIBS CACHE BOOL OFF )
```

The build is a matter of issuing the make command when the cmake process completes successfully. The subsequent testing can be considered unit testing as it happens “in-build”. It has to be launched on the compute nodes and the following PBS script is used:

```
#!/bin/bash --login
#PBS -N ECKitTest_n32
#PBS -l mppwidth=32
#PBS -l mppnppn=32
#PBS -l walltime=00:20:00
#PBS -A y07

module load CMake/2.8.8
module list

cd $PBS_O_WORKDIR
echo "ECKIT In build Test"
aprun -n 1 -d32 -cc 0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30 -a xt -b make -j 16 test >
${PBS_JOBID}.out
```

See Appendix C for the tests and their results

7.2.2.4 FDB libraries

Similar to the ECKIT build, the makefiles for FDB are created with CMake with the following options:

Figure A4: Generating the makefiles for FDB

```
export LD_LIBRARY_PATH=/work/y07/y07/oifs/install/grib_api/1.9.18/cce_8.1.8-
il/lib:${LD_LIBRARY_PATH}

cmake .. \
-DCMAKE_MODULE_PATH=/home/y07/y07/oifs/ecmwf/ecbuild/cmake \
-DCMAKE_C_COMPILER=cc \
-DCMAKE_CXX_COMPILER=CC \
-DFDB_FORTRAN_API=OFF \
-DCMAKE_Fortran_COMPILER=ftn \
-DCMAKE_BUILD_TYPE=Release \
-DENABLE_RPATHS=OFF \
-DCMAKE_INSTALL_PREFIX=/work/y07/y07/oifs/install/fdb/5.0.0/cce_8.1.8-il \
-DBUILD_SHARED_LIBS=OFF \
-DGRIB_API_PNG=OFF \
-DGRIB_API_JPG=OFF \
-DCMAKE_PREFIX_PATH="/work/y07/y07/oifs/install/eckit/0.3.0/cce_8.1.8-il;\
/work/y07/y07/oifs/install/grib_api/1.9.18/cce_8.1.8-il"
```

A toolchain file is used for the cross-compilation. This is because the compilation occurs on the login nodes that are AMD Istanbul and do not understand some of the extra instructions for Interlagos processors. A program compiled for the compute node will exit with “illegal instruction” which is the same as “SIGILL” i.e. an error status of -4. The tests are (simulated) during the build. This is risky as it gives the impression that those tests succeeded even though they had not been exercised. They have to be processed later by a job script on the compute nodes to confirm the results.

7.2.3 Add a module to activate the libraries for users.

Three module files were created so that the CCE version of the libraries can be used more easily when building OpenIFS.

Figure A5 detail of fdb_cce module 0.0.2

```
#%Module#####################################################################
#
# ECMWF database library FDB
#
proc ModulesHelp { } {
puts stderr "FDB 5.0.0 - for use with OpenIFS"
puts stderr "This module is to set up the CCE environment"
puts stderr " Requires version 8.1.8 of CCE "
puts stderr "Installer: M. Richardson, NAG"
puts stderr "Date: 12/05/13 Updated 17th July 2013"
}
conflict PrgEnv-gnu
conflict PrgEnv-pgi
conflict PrgEnv-pathscale
prereq cce/8.1.8
#
#  the following are therefore parameterised and generic
#
set GRIB_VERSION 1.9.18
set ECKIT_VERSION 0.3.0
set FDB_VERSION 5.0.0
set OIFS_PKGS /work/y07/y07/oifs/install
# test for which compiler level
set PLATFORM cce_8.1.8-il
      puts stderr " Open IFS support libraries installed at $OIFS_PKGS"
      puts stderr " Platform is $PLATFORM"
      puts stderr " Version of library FDB is $FDB_VERSION"
      puts stderr " Version of library ECKIT is $ECKIT_VERSION"
      puts stderr " Version of library GRIB_API is $GRIB_VERSION"
#
set  GRIB_API_ROOT $OIFS_PKGS/grib_api/$GRIB_VERSION/$PLATFORM
setenv  GRIB_API_PATH $OIFS_PKGS/grib_api/$GRIB_VERSION/$PLATFORM
prepend-path PATH $GRIB_API_ROOT/bin
prepend-path LD_LIBRARY_PATH $GRIB_API_ROOT/lib
setenv  GRIB_DEFINITION_PATH $GRIB_API_ROOT/share/grib_api/definitions
#
set     ECKIT_ROOT $OIFS_PKGS/eckit/$ECKIT_VERSION/$PLATFORM
setenv  ECKIT_PATH $OIFS_PKGS/eckit/$ECKIT_VERSION/$PLATFORM
prepend-path LD_LIBRARY_PATH $ECKIT_ROOT/lib
#
# CAUTION fdb_root here is a local variable for the tcsh script
set FDB_ROOT ${OIFS_PKGS}/fdb/${FDB_VERSION}/${PLATFORM}
setenv  FDB_PATH ${OIFS_PKGS}/fdb/${FDB_VERSION}/${PLATFORM}
prepend-path PATH ${FDB_ROOT}/bin
prepend-path LD_LIBRARY_PATH ${FDB_ROOT}/lib
#
module-whatis   "ECMWF library FDB on HECToR Phase 3"
```

The module files have been placed in the modules sub-directory of the "oifs" account. A user will access them with "module use ~oifs/modules"

module load grib_api_cce to specificly set the GRIB_API environment
module load fdb_cce to specify the FDB environment

The openifs module invokes the grib_api and fdb modules but those are available for other software if needed separately.

module load openifs_cce to setup the OpenIFS environment

The numbering was started at 0.0.1 for the first and subsequent developments are indicated by incrementing the third "revision" digit. The working version was established with 0.0.2 that adds associates the correct version of grib_api_cce and fdb_cce to openifs_cce. More recent developments have been encapsulated in version 0.0.3. It is good practice to interrogate the modules to determine the prevailing default ("module avail openifs_cce").

FIGURE A5: detail of module activation

```
% module use ~oifs/modules
% module show openifs_cce
-------------------------------------------------------------------
/usr/local/packages/oifs/modules/openifs_cce/0.0.3:

module-whatis    CCE versions of support libraries to build OpenIFS
conflict         PrgEnv-gnu
conflict         PrgEnv-pgi
prereq   cce/8.1.8
module           load grib_api_cce/0.0.3
module           load fdb_cce/0.0.3
prepend-path     PATH /work/n02/n02/hum/fcm/bin
setenv           OIFS_IFSDATA /work/y07/y07/oifs/data/ecmwf/ifsdata
setenv           OIFS_ARCH x86_64
setenv           OIFS_COMP cce_fdb
setenv           OIFS_BUILD opt
```

7.3 Testing the installations

There is a set of tests for each package, the tests and results are detailed in appendix C. As detailed above there is cross-compilation being processed so the actual tests were arranged to occur on the compute node (this is part of the reason for installing the libraries in the WORK partition as sown in table A1.

Table A1: The locations of the installed libraries	
GRIB_API	/work/y07/y07/oifs/install/grib_api/1.9.18/cce_8.1.8-il
ECBUILD	/home/y07/y07/oifs/ecmwf/ecbuild
ECKIT	/work/y07/y07/oifs/install/eckit/0.3.0/cce_8.1.8-il
FDB	/work/y07/y07/oifs/install/fdb/**5.0.0pre**/cce_8.1.8-il
MARS	/**home**/y07/y07/oifs/mars/0.3.0/cce_8.1.8-il

7.3.1 FDB

The "in-build" tests that have been deferred are run using a PBS script. To do this simply, it is easier to build the software within the LUSTRE file system and then be able to run the compute node. [Jul2013 it is now possible to access $HOME from CNL]

NOTE the FDB_HOME and FDB_ROOT must be set (to the same location) where the data files are to be written.

Here is the PBS script:

```
#!/bin/bash --login
#PBS -N FDB_BuildTest
#PBS -l mppwidth=32
#PBS -l mppnppn=32
#PBS -l walltime=00:20:00
#PBS -A y07

module swap cce cce/8.1.8
module load CMake/2.8.8
module list

cd $PBS_O_WORKDIR
export FDB_ROOT=$PBS_O_WORKDIR
export FDB_HOME=$PBS_O_WORKDIR
export TMPDIR=$PBS_O_WORKDIR/tmp

OIFS_PKGS=/work/y07/y07/oifs/install
GRIB_VERSION=1.9.18
ECKIT_VERSION=0.3.0
PLATFORM=cce_8.1.8-il

export GRIB_API_PATH=$OIFS_PKGS/grib_api/$GRIB_VERSION/${PLATFORM}
export GRIB_DEFINITION_PATH=$GRIB_API_PATH/share/grib_api/definitions
export GRIB_SAMPLES_PATH=$GRIB_API_PATH/share/grib_api/samples
#
export PATH=$GRIB_API_PATH/bin:${PATH}
export LD_LIBRARY_PATH=$GRIB_API_PATH/lib:${LD_LIBRARY_PATH}
#
export ECKIT_PATH=${OIFS_PKGS}/eckit/${ECKIT_VERSION}/${PLATFORM}
export PATH=${ECKIT_PATH}/bin:${PATH}
export LD_LIBRARY_PATH=${ECKIT_PATH}/lib:${LD_LIBRARY_PATH}
#
echo "FDB build and Test"
echo "LD_LIBRARY_PATH is" $LD_LIBRARY_PATH
echo "++++++++++++++++++++++++++++++++++++++++++++"
set -x
aprun -n 1 -d 32 -a xt make -j 16 test
```

7.3.2 Testing the software after installation.

The installation is a matter of issuing the “make install” command. Subsequent testing can be done with provided tests. For FDB there are two post-install checks: testa01 and testa02. See appendix C for the tests and their results.

7.4 Installation of a MARS client

MARS is a method of interacting with numerical weather forecasting data and formatting it for use in other tools. One use is from within Metview and another is as a standalone client to validate the builds of the support libraries. It is built for the login nodes and thus the environment has to be switched from the default settings to a GNU compiler and set the platform for Istanbul processors.

module swap PrgEnv-cray PrgEnv-gnu
module swap xtpe-interlagos xtpe-istanbul

A preliminary build was achieved on 26 April 2013 under the guidance of ECMWF staff.

The mars build script generates a site specific file (the assumption is the build is not a cross-compilation).

-rw-r----- 1 oifs oifs 1008 2013-04-26 11:55 config.linux.2.site-noFDB

The source is supplied by ECMWF as a tar file: mars_client_grib_api.20110523.tar.gz
It is unpacked in ~oifs/MARS_client/src/mars_client_grib_api
Environment is set as:
GCC 4.7.2
ARCH=linux
It can be guessed from $platform which is built from
uname –m ($machine) and uname –s ($os)
Platform="$machine:$os"
Config.linux.2.cfg
There are 3 sub-directories: chk, etc, tools they were not modified.
Only the config.linux.2.site was changed to add the FDB instead of NOFDB.
(Compared to config.linux.2)
LOCAL_CFLAGS= -DR64 -DFDB
Mars_build is an interactive script to prepare the build; it asks several questions and then processes the code. It will create/overwrite config.platform.site. The second build included the FDB flag and thus linked to a version of FDb built with GCC 4.7.2 and target cpu "Istanbul".
Files modified that are not object files:
config.linux.2.site-noFDB my saved version of the site config
mars.sh this sets several environment variables for use with MARS
make.linux.2.site the ld command to create an SO from a static archive.
make.dep empty file to force a full build (check Makefile)
config.linux.2.site the site specific configuration (generated by the mars_build script).
rpcmars.h an "rpcgen" file
marsxdr.c something for external data representation (Where/why what)
langl.c definition of system specifc parameters e.g. INT32_MIN
langy.c some sort of YACC director
libmars.a the mars static library for the client and other s/w
marsthe mars client
For HECToR and "oifs" this package was installed in:
~oifs/MARS_client/0.0.1/gcc_4.7.2-mc12
The MARS development team provided a basic test case in which a GRIB2 file is processed to extract data and compared to a reference set. Currently there is no FDB-MARS test.

8 Appendix B: Results of post-build testing

8.1 B. 1. Result of tests for the CCE build of GRIB_API

```
PASS: definitions.sh
PASS: ieee.sh
PASS: grib1to2.sh
PASS: grib2to1.sh
PASS: badgrib.sh
PASS: ls.sh
PASS: convert.sh
PASS: filter.sh
TEST: ./multi.sh : no data to test
PASS: multi.sh
PASS: budg.sh
PASS: gridType.sh
PASS: concept.sh
PASS: decimalPrecision.sh
PASS: bitsPerValue.sh
PASS: get_fail.sh
PASS: missing.sh
PASS: local.sh
PASS: step.sh
PASS: set.sh
PASS: iterator.sh
PASS: compare.sh
PASS: level.sh
PASS: index.sh
PASS: bitmap.sh
PASS: list.sh
PASS: second_order.sh
TEST:./multi_from_message.sh: no data to test
PASS: multi_from_message.sh
PASS: change_scanning.sh
PASS: julian.sh
PASS: statistics.sh
PASS: tigge.sh
PASS: tigge_conversions.sh
TEST: ./read_any.sh : no data to test
PASS: read_any.sh
PASS: padding.sh
PASS: debug.sh
==================
All 35 tests passed
==================
PASS: copy_message.sh
PASS: get.sh
PASS: get_data.sh
PASS: get_pl.sh
PASS: get_pv.sh
PASS: keys_iterator.sh
PASS: nearest.sh
PASS: precision.sh
PASS: multi_write.sh
PASS: multi.sh
PASS: print_data.sh
PASS: set.sh
PASS: set_bitmap.sh
PASS: set_missing.sh
PASS: set_pv.sh
PASS: samples.sh
PASS: count_messages.sh
PASS: read_message.sh
PASS: index.sh
==================
All 19 tests passed
==================
PASS: iterator.sh
PASS: get.sh
PASS: print_data.sh
PASS: set.sh
PASS: keys_iterator.sh
TEST: ./multi.sh
SKIP: ./multi.sh
PASS: multi.sh
PASS: multi_write.sh
PASS: precision.sh
PASS: list.sh
==================
All 9 tests passed
=====
```

There were no Fortran tests and no python tests attempted.

8.2 B.2. Testing the CCE build of ECKIT

```
Running tests...
Test project /work/y07/y07/oifs/build/eckit_0.3.0-cce_8.1.8-il
    Start  1: test_config
 1/13 Test  #1: test_config .....................   Passed    0.07 sec
    Start  2: test_resource
 2/13 Test  #2: test_resource ....................   Passed    0.15 sec
    Start  3: test_log
 3/13 Test  #3: test_log .........................   Passed    0.16 sec
    Start  4: test_log_channels
 4/13 Test  #4: test_log_channels .................   Passed    0.16 sec
    Start  5: test_log_callback
 5/13 Test  #5: test_log_callback .................   Passed    0.15 sec
    Start  6: test_log_threads
 6/13 Test  #6: test_log_threads .................***Exception: Other  0.16 sec
    Start  7: test_aiohandle
 7/13 Test  #7: test_aiohandle ....................***Failed    0.18 sec
    Start  8: test_mutex
 8/13 Test  #8: test_mutex ........................   Passed    0.15 sec
    Start  9: test_string_tools
 9/13 Test  #9: test_string_tools .................   Passed    0.15 sec
    Start 10: test_tokenizer
10/13 Test #10: test_tokenizer ....................   Passed    0.16 sec
    Start 11: test_cache
11/13 Test #11: test_cache ........................   Passed    0.15 sec
    Start 12: test_producer
12/13 Test #12: test_producer .....................***Exception: Other  0.17 sec
    Start 13: test_context
13/13 Test #13: test_context ......................   Passed    0.16 sec

77% tests passed, 3 tests failed out of 13

Total Test time (real) =   2.15 sec

The following tests FAILED:
        6 - test_log_threads (OTHER_FAULT)
        7 - test_aiohandle (Failed)
       12 - test_producer (OTHER_FAULT)
```

The three failed tests have been noted but ignored on the advice from ECMWF.

8.3 B.3. Testing the CCE build of FDB

```
FDB build and Test
LD_LIBRARY_PATH is /work/y07/y07/oifs/install/eckit/0.3.0/cce_8.1.8-
il/lib:/work/y07/y07/oifs/install/grib_api/1.9.18/cce_8.1.8-il/lib:
++++++++++++++++++++++++++++++++++++++++++++
Running tests...
Test project /work/y07/y07/oifs/build/fdb_5.0.0-cce_8.1.8-il
    Start 1: test_fdb_params
1/9 Test #1: test_fdb_params ..................   Passed    0.05 sec
    Start 2: test_fdb_index
2/9 Test #2: test_fdb_index ...................   Passed    6.83 sec
    Start 3: test_fdb_toc
3/9 Test #3: test_fdb_toc .....................   Passed    6.19 sec
    Start 4: test_fdb_agentwriter
4/9 Test #4: test_fdb_agentwriter .............   Passed   12.57 sec
    Start 5: test_fdb_agentreader
5/9 Test #5: test_fdb_agentreader .............   Passed   40.35 sec
    Start 6: test_fdb_service_1_write
6/9 Test #6: test_fdb_service_1_write .........   Passed   24.35 sec
    Start 7: test_fdb_c_api
7/9 Test #7: test_fdb_c_api ...................   Passed   57.00 sec
    Start 8: atest01
8/9 Test #8: atest01 ..........................   Passed   11.10 sec
    Start 9: atest02
9/9 Test #9: atest02 ..........................   Passed    0.74 sec

100% tests passed, 0 tests failed out of 9

Total Test time (real) = 168.95 sec
```

9 Appendix C: Using OpenIFS on HECToR

The module openifs_cce is used to set up the environment.

```
% module show openifs_cce
-------------------------------------------------------------------
/usr/local/packages/oifs/modules/openifs_cce/0.0.3:

module-whatis    CCE versions of support libraires to build OpenIFS
conflict         PrgEnv-gnu
conflict         PrgEnv-pgi
prereq   cce/8.1.8
module           load grib_api_cce/0.0.3
module           load fdb_cce/0.0.3
prepend-path     PATH /work/n02/n02/hum/fcm/bin
setenv           OIFS_IFSDATA /work/y07/y07/oifs/data/ecmwf/ifsdata
setenv           OIFS_ARCH x86_64
setenv           OIFS_COMP cce_fdb
setenv           OIFS_BUILD opt
-------------------------------------------------------------------
```

9.1 Building OpenIFS

The modules system is helpful for setting the correct environment for bulding OpenIFS. i.e. module use ~oifs/modules, module load openifs_cce .
Currently the default module is 0.0.3 but the initial work was done using version 0.0.2 of the module. this module sets the following environment variables

OIFS_IFSDATA /work/y07/y07/oifs/data/ecmwf/ifsdata
OIFS_ARCH x86_64
OIFS_COMP cce_fdb
OIFS_BUILD opt

and the PATH so that the “fcm” tool is available for building OpenIFS with the command:

fcm make -j 4 -vv --new -f cfg/oifs.cfg

The result is a binary (master.exe) that can be launched from within a “standardized” script named **ifs_run.sh**. The configuration file “oifs.cfg” contains the compilation and linking options for OpenIFS some additional changes were made to use Huge Page Sizes (see section 5).

Three test cases were provided for initial baseline testing:

t159/2012102700
t511/2012102700
t1279/2012102700

Later changes to the source were needed to run a productin simulation T1279_op and the following changes were needed as shownin Figure B1.

Figure B1: GRIB Code changes
NFPPHY=86,
MFPPHY=31,32,33,34,35,36,37,38,39,40,41,42,44,45,49,50,57,58,59,78,79,129,136,137,139,141,142,143,144,145,146,147,151,159,164,165,166,167,168,169,170,172,175,176,177,178,179,180,181,182,183,186,187,188,189,195,196,197,198,201,202,205,206,208,209,210,211,235,236,238,243,244,245,229,230,231,232,213,212,8,9,228089,228090,228001,260121,260123,

9.2 Operation without FDB

This is the default operation of OpenIFS. There are two GRIB files produced on each specified time step and the files are named accordingly. One contains grid point data and the other contains spectral data.
For example with the T159 case the following were generated.

ICM**GG**fwj8+000000 (14.5MB), ICM**GG**fwj8+000012, ICM**SH**fwj8+000000 (32MB), ICM**SHf**wj8+000012, ICM**SH**fwj8+000108, ICM**SH**fwj8+000120

9.3 Operation with FDB

Instead of generating "per time step" GRIB files in the case directory, a subdirectory is named "fdb/<fdbxyz123>" and the FDB system creates a fresh sub-directory named after the RUNID and experiment type.

Then a number of files are created in groups of 3. One DAT file, an IDX file and a FILES file. The DAT file is a GRIB format concatenation of more than one GRIB "message". Note there are as many DAT files as there are "writer tasks" and by default this is equal to the number of MPI tasks being used for the simulation.

Figure B2: files created by the FDB process in the fdb sub-sub-directory

Size in bytes	File name
165953536	0000:fc:sfc.0.20131016.084031.7773890805760.dat
65536	0000:fc:sfc.0.20131016.084031.7773890805761.idx
267	0000:fc:sfc.0.20131016.084031.7773890805761.idx.files
165953536	0000:fc:sfc.0.20131019.164347.88669599825920.dat

(an extra time stamp layer is created between the case directory and the fdb data.The number of files is determined by the number of fields so if many MPI tasks are available there may be fewer files. The number of writers can be varied and this is investigated later in the report.

The repeat of the run will generate a new set of FDB dat files witht a fresh timestap name. in the figure B2 the later run on 19th October is shown along with the earlier 16th October run.