Parallelisation of CABARET

Phil Ridley
Numerical Algorithms Group Ltd,
Wilkinson House, Jordan Hill Road,
Oxford, OX2 8DR, UK,
`email: phil.ridley@nag.co.uk`

February 1, 2011

Abstract

The CABARET (Compact Accurate Boundary Adjusting high REsolution Technique) code may be used to solve the compressible Navier-Stokes equations. The code is based on a low dissipative and low dispersive conservative CABARET method that constitutes a substantial upgrade of the second-order upwind leapfrog scheme. Most notably, the CABARET algorithm has a very local computational stencil that for scalar advection constitutes only one cell in space and time.

The present application of CABARET is for the investigation of aircraft noise, which is currently a very important environmental concern. An important component of aircraft noise is due to airframe/engine installation effects, the reduction of this remains a very challenging problem. In particular, when deployed at a large angle of attack at approach conditions, the wing flaps become a very important noise source. For engine-under-a-wing configurations, flap interaction with the jet can even become a dominant noise component. A crucial element of any noise prediction scheme is the high fidelity Large Eddy simulation (LES) model. For the airframe/engine noise problem, this model needs to accurately capture all important wing-flap, free jet and wing-flap-jet interaction effects.

With the high-resolution CABARET algorithm capable of resolving the fine-flow structures on coarse grids, we have already reduced the problem size needed for acoustic-sensitive modelling. But for high-fidelity LES modelling, we require a computational means capable of handling grid resolutions of the order of, at least, several millions of points. Hence for this dCSE project we will develop a scalable CABARET code enabling use of the full capability of HECToR for investigating grid sensitivity effects on the flow transition to turbulence in the initial part of the jet as well as to study the effect of inflow conditions, e.g., with and without including the nozzle exit geometry in the calculation.

This project was proposed by Dr Sergey Karabasov of the University of Cambridge Engineering Department and was approved for 12 months effort at the December 2008 round. The project began in March 2009 and was completed January 2011 after 10 months full time effort. For jet-flap-interaction, this dCSE will provide a key computational tool that will hopefully help to answer the question "What is the effect of fine-scale-flow structures on far-field noise in the audible range of frequencies?"

Contents

1 Introduction

1.1 Application Background

A large number of high resolution numerical schemes exist for solving general CFD problems. Typically, those are either more suitable for either shock capturing or linear wave propagation. General purpose high-resolution schemes also exist but in many of them additional implementation and computational costs arise from the boundary conditions because of the stencil complexity and numerical robustness.

Robust general-purpose high-resolution numerical algorithms are particularly valuable in the modelling of turbulent flows, especially in the context of Large Eddy Simulations (LES). LES relies on the ability of the numerical method to resolve all flow scales above a certain threshold and those which are below this threshold are removed from the solution by using either a numerical realisation of ones favorite closure model of turbulent dissipation or implicitly, by a numerical dissipation. For the latter, one efficient strategy is to use numerical flux-correction methods in the framework of a Monotonically Integrated LES (MILES) approach. For the MILES component of this work, the Compact Accurately Boundary Adjusting High REsolution Technique (CABARET) is used. This is because the CABARET method is a cutting-edge numerical scheme for hyperbolic-type equations that has already been successfully applied for solving a range of convection/advection-dominated problems [1]–[5].

1.2 Why Use CABARET?

CABARET is a general-purpose advection scheme which is suited for computational aeronautics and geophysics problems. For solving Navier-Stokes equations with Reynolds numbers of 10^4, the method gives a very good convergence without any additional preconditioning down to Mach numbers as low as $0.05 - 0.1$. In particular for the MILES modelling of a hydrodynamic instability and free jet, a 257^2 grid using CABARET is able to produce results comparable to a conventional second order method which would require at least 1025^2 grid points [5]. Here, the CABARET method is 30 times more efficient.

For linear advection, the CABARET scheme is a modification of the non-dissipative and low-dispersive Second-order Upwind Leapfrog method [6]. The modification consists of introducing separate conservation and flux variables that are staggered in space and time this results in a very compact, one cell in space and time computational stencil. In comparison to the standard finite-difference and finite-volume methods, in CABARET there is always an additional independent evolutionary variable, which gives the method the ability to preserve one more important property of the governing equations - the small phase and amplitude error. Traditionally conservation fluxes are computed at cell faces using cell-centre variable interpolation but with CABARET the conservation fluxes explicitly depend on the evolution of cell-centre and cell-face variables. For enforcing the non-oscillatory property of the solution, the CABARET scheme uses a low-dissipative non-linear flux correction that is directly based on the maximum principle for the flux variables.

The key role of the nonlinear correction in the MILES CABARET method is to remove the under-resolved fine scales from the solution so that an accurate balance between the numerical dissipation and dispersion errors is preserved. For the Navier-Stokes equations, each time iteration of the CABARET method consists of a conservation phase and a characteristic decomposition phase.

1.3 Outline of this project

The CABARET method can be used on structured and unstructured grids. To deal with high-fidelity simulations requiring the use of unstructured grids, a version of the CABARET code has been developed. This is capable of dealing with an arbitrary grid structure and will be the code that is used for this project. At the start of this project, the CABARET code was mainly used on single core desktop systems.

We shall describe the development of a distributed memory version of CABARET for use on the HECToR XT/XE systems and based on an unstructured grid representation for compressible turbulent flows. It is worthwhile noting that relationships between neighbouring cells and partitions are generally more complex to manage here than with a structured grid. The consequence is that data associated with unstructured grid layouts requires a lot of effort so that it can be managed and updated by the methods which determine new values based on other values in their physical proximity.

The original work plan for this project was scheduled as follows:

1. Develop an automatic geometrical domain decomposition for parallel processing, which should balance the loads efficiently in relation to the CABARET hexahedral grid structure. This will be implemented by a pre-process stage of the data from the Gambit mesh generator facilitating use of either METIS or Scotch. Produce an internal report on the best partitioning method and demonstrate effectiveness for at least a million grid points.
2. Implement a new MPI parallel version of CABARET using data passing protocols between internal boundaries (cell faces and sides).
3. Validate and test the new code using a 3D backward-facing step case while expecting 70% parallel efficiency when using 256 cores of the XT4 part of HECToR with the Phase 2a architecture.
4. Demonstrate that the new code works for at least 50 million grid points on the XT4 part of HECToR, using the 3D backward-facing step case.

This work began March 2009 with the end goal being a scalable parallel code which would facilitate much larger simulations than possible with the serial code. By implementing MPI, the data would be located over many distributed processing nodes on HECToR using optimum communication that would enable the application to run simulations at least 100 times larger than previously and with a much faster turn around time.

This report will discuss the development of the CABARET code in relation to:

– The partitioning method for the data decomposition.
– Compiler performance.
– Efficient use of hybrid parallelisation using MPI and OpenMP.

2 CABARET Implementation for HECToR

2.1 Introduction

The serial version of the CABARET code emerged as the result of more than 12 years work of Dr Karabasov and his collaborators within the group of Prof. Goloviznin of the Russian Academy of Science, Moscow. This code has been used extensively for grids involving up to 10^5 points for both compressible and incompressible flows, however current state of the art high fidelity MILES simulations require millions of grid points. This could be achieved with a new version of the code, re-designed for distributed processing. CABARET is particularly suited to this approach since the computational stencil only requires information from the first layer of nearest neighbour cells.

2.2 Code structure

The existing CABARET code was around 4000 lines of Fortran 90, so this was used as an outline for the parallel version. The code included routines for; reading in the computational grid, reading from /writing to restart files, data output to Tecplot format, as well as the main computational part which can be separated into the following five steps:

PHASE1 - This is the conservative predictor step. This calculation updates the neighbouring conservative type variables. Interpolation of the cell based variables to the grid points is also performed.

VISCOSITY - Computation of the cell centred viscous terms. This step updates the cell based variables.

PHASE2 - Extrapolation step where the local cell-based characteristic splitting is performed. This is where the most intensive computation lies.

BOUND - Applying conditions for the boundary cells. This determines the new values of the flux-type cell face values and only involves the physical boundary.

PHASE3 - The conservative corrector step. This also updates the cell based variables with their values from the previous time step.

2.3 Input Grid

The computational grid for CABARET is a Gambit(Fluent) generated unstructured linear hexahedral mesh. This is described in an ASCII file which consists of; the total number of apexes(NAPEX), cells(NCELL), sides(NSIDE) and boundary sets(NBSETS); a list of the sides for each cell - GEMCELLSIDE; a list of the cell connectivity for each side - GEMSIDECELL; a list of the cartesian vertices - APEX; a list of the vertex connectivity for each cell - GEMCELLAPEX; a list of the sides for each boundary set - IBOUND. For a 10^5 cell mesh, this file is around 33MB and for a 5.12×10^7 cell mesh, around 17GB.

2.4 Output Data

The output data is produced at intervals throughout a simulation which are pre-determined by the user. There are two data files produced; for restarts and Tecplot visualisation. The restart data is written to an unformatted file and contains mesh parameters along with the current flux and conservative variable values. The Tecplot data file is written as ASCII. This file also contains flux and conservative variable values along with mesh data and local vortex identification criteria - the Q criterion. For a 10^5 cell mesh, the restart data file is around 15MB and for a 5.12×10^7 cell mesh, around 8GB. The Tecplot file is at least double this size and can be larger depending upon how many checkpoints are chosen by the user.

2.5 Implementing Data Parallelism

For the computation, we can summarise that CABARET involves calculations based on variables which either relate to the flux type, cell sides (SIDE) or the conservative type cell based (CELL). Additionally there is also the grid point based variables (APEX) which are calculated from the neighbouring CELL values.

In terms of implementing data parallelism within the code, the CABARET algorithm requires knowledge of only the nearest neighbouring cells, so we need to store a set of halo values of CELL, SIDE and APEX on each process and then update them at appropriate points in the calculation. This is in addition to the local to global address mapping information regarding the computational grid. However, this remains static throughout the calculation and so no communication is required for this data.

The values in the APEX array require no communication since they can always be determined from the halo CELL values and local process CELL values. Whenever there is an update to the flux type i.e. SIDE values we shall need to communicate updates to the halo CELL values on the other neighbouring processes. This is because each SIDE has two associated CELLs as shown in Figure 1 and both of these may not necessarily exist as halo CELLs on each process.

This aspect of the computation means that we need updates for halo values of SIDE based calculations to be communicated both before **PHASE2** and after **BOUND** for each time step of the CABARET calculation. The actual implementation is described below and this includes two nearest neighbour based communications which involve updating both the conservative and flux type (CELL and SIDE) variables.

PHASE1 - Calculate new cell (conservative) based variables and communicate them between nearest neighbours for updating the neighbouring conservative type variables (CELL). Interpolate the cell based variables to the grid points.

VISCOSITY - Update new cell based variables.

PHASE2 - Prior to the main (i.e. extrapolation) computation for SIDE, communicate the flux type cell face variables along with the cell based variables between nearest neighbours, i.e. SIDE and CELL.

BOUND - Apply boundary conditions. Calculate new values of the flux type cell face values (SIDE).

PHASE3 - Update the cell based variables and perform a global reduction to determine the size of the next time step.

To summarise the computational process, which is repeated at each time step. This includes two nearest neighbour based communications and one global reduction. This was implemented within the CABARET code with MPI, using a local (indirect) referencing system for the APEX, SIDE, and CELL data for each MPI process and non-blocking MPI calls to update halo data at each time step. When output data is required, (for restarts and Tecplot) data is sent to the master process for writing to two separate files for post processing.

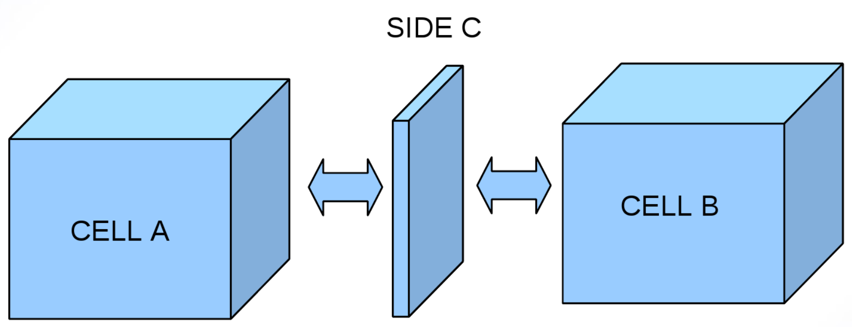

Figure 1: A side with two neighbouring cells.

2.6 Data decomposition

The numerical domain decomposition is described as a Gambit (Fluent) unstructured hexahedral cell mesh, however, the geometrical arrangement in question may allow for a regular data decomposition for parallelisation. Unstructured codes such as CABARET have a potentially high degree of instruction parallelism, but their data accesses requires a large amount of indirection, which increases memory operations for the indexing information and usually results in poor spatial locality and higher latencies for data access. This is not to say that the underlying algorithms will not scale as efficiently to as many cores as a structured grid based scheme would.

Unstructured grids require a list of the connectivity which specifies the way that a given set of apexes form the individual cells (indirect referencing). This is not required in a structured approach since we can usually deal with the local to global references via offsets. The performance of the parallel CABARET code will rely heavily upon an efficient data partitioning. Both in terms of performing the actual decomposition itself and also giving a well load balanced and communication optimized simulation.

Algorithms for finding efficient partitions of unstructured grids must ensure that the number of cells assigned to each processor is roughly the same, and also that the number of adjacent cells assigned to different processors is minimised. The goal of the first condition is to achieve efficient load balancing for the computations among the processors. The second condition ensures that the communication pattern resulting from the placement of adjacent cells to different processors can be optimised.

2.7 Partitioning the Unstructured grid

To partition the unstructured Gambit mesh we need to assign certain cells to appropriate MPI processes in the communicator. This depends upon the number of MPI processes that one wishes to run the end simulation with. One suitable approach is to use an external partitioning package such as METIS [7], [9] or Scotch [8], [10]. Alternatively, we may be able to use a more simple method of structured partitioning across one, two or three dimensions of the cartesian plane.

The choice of whether to use an external partitioning package or a structured decomposition depends upon the geometry in question. For our test case within this project, (backward facing step geometry) subsequent results will show that it is most efficient to use a structured decomposition, however, for the arbitrary geometries that may be considered in future CABARET simulations it is necessary for the code to have the ability to handle unstructured decompositions.

The communication routines within the code have been designed to accomodate unstructured decompositions, if required. Also, a separate (pre-process) partitioning routine has been developed. This routine can be used to process a decomposition from an external package or generate a structured one. Due to the unstructured aspect of the Gambit grid, there will be non-contiguous data. The partitioner has to either have enough global memory allocated to read in the entire grid or else in separate parts. However, it is more efficient to read in the entire grid on a single XT/XE node and then use a shared memory approach.

Therefore the partitioner has been implemented with hybrid OpenMP/MPI parallelism for multi-core efficiency and the end application is capable of generating the partition data for a 5.12×10^7 cell Gambit mesh for 250 MPI partitions within 29s wct (using 240 cores over 10 fully populated Phase 2b nodes (XT6 - 24 core Opteron 6172 2.1GHz processors, arranged as two dual hex core sockets per node in a non uniform memory architecture). It can be run with

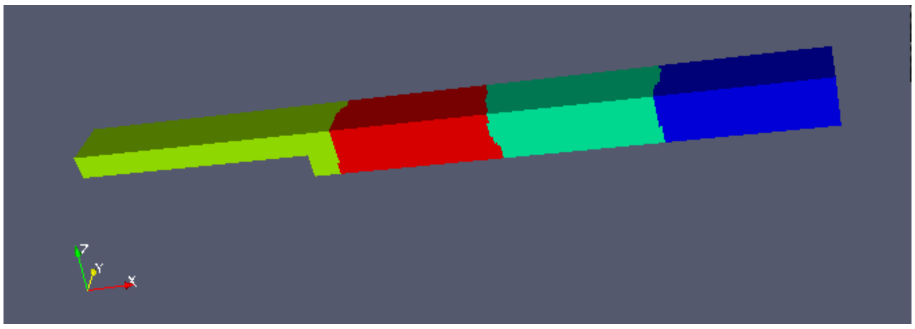

Figure 2: Simple unstructured decomposition with four partitions.

up to the same number of cores as requested partitions. It exhibits linearly scalability as the algorithm involves little communication.

2.8 Performance of the code

To benchmark initial performance of the MPI only CABARET, the wall clock time to perform 276 time steps without any I/O was measured against the performance of the serial code to perform the same task. The test case used here is a 3D backward facing step geometry and the boundary conditions are for laminar flow, with Reynolds number=5000 and Mach number=0.1. The grid size is 10^5 hexahedral cells. Both codes were compiled and run on Phase 2a of HECToR (with each node a quad core 2.3GHz Opteron processor with 8GB of RAM) using the PGI 10.8.0 Fortran90 compiler. The time taken for the serial code was 360 seconds and the parallel code 7.8 seconds using 50 cores (13 nodes) and 2 seconds using 250 cores (63 nodes). This gives an effective parallel efficiency of 72% for 250 full populated nodes.

3 Efficiency of Partitioning Methods

3.1 Comparison of the Efficiency of Partitioners

Due to the unstructured grid used within the main CABARET algorithm, the choice of whether to use an external partitioning package or a structured decomposition depends upon the geometry in question. Wherever possible it is recommended to use a structured approach based on partitioning along the cartesian planes. If however, the geometrical arrangement is more complex where it is not possible to obtain a reasonable (structured) partition which gives a well load balanced problem and optimum communication arrangement, then an unstructured partitioning algorithm is required such as METIS [7], [9] or Scotch [8], [10]. The parallel CABARET code has been designed so that it is capable of running with either a structured or unstructured data decomposition.

The METIS partitioner uses a multi-level k-way approach [12] whereas the Scotch package employs a dual recursive bipartitioning approach [11]. In terms of performance, for the time taken to partition a 10^5 cell backward facing step geometry, both METIS and Scotch give similar results, when producing 32, 50, 100, and 250 partitions. For this test case Scotch does give slightly better performance - a 5% reduction in CABARET run time. But METIS is slightly easier to use since it does allow the user to provide it with input which is close to that of the original unstructured grid whereas Scotch requires the input to be in the form of a dual graph. Figure 2 illustrates how a 3D backward facing step geometry is partitioned with an unstructured partitioner.

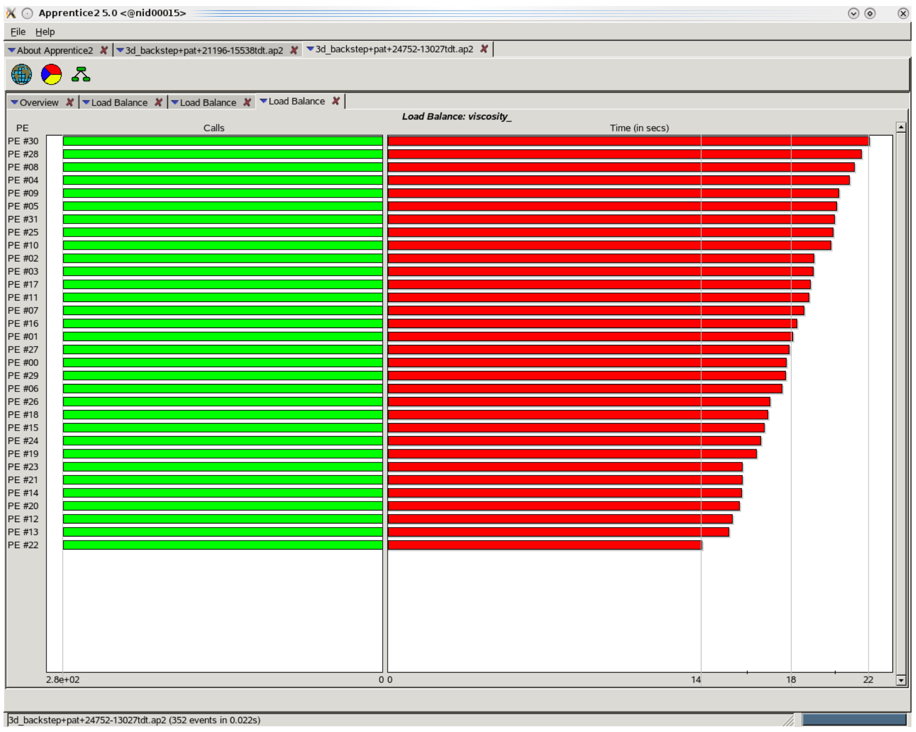

Figure 3: Uneven load balance for the **VISCOSITY** calculation.

3.2 Partitioning and Load Balancing

The main reason why partitioning the grid with Scotch enables CABARET to run slightly better is that for this particular problem, the partitioning produces more efficient load balancing and a balanced communication. The CrayPat profile in Figure 3 demonstrates what often happens when using an unstructured partition for a structured geometry, e.g. Scotch or METIS. Here, there are 32 MPI processes (partitions) and process number 30 appears to have a third more work than process 22. The problem is caused by an uneven balance of cells within each partition along with the number of cells in each partition which contain boundary faces. The calculations which are dependent upon the boundary cells are **VISCOSITY** and **BOUND**. If there is an unfair distribution of the boundaries within the partitions then certain partitions may have an uneven load balance to cope with and this will in turn limit scalability.

For a structured geometry such as the 3D backward facing step, it is more efficient to use a structured partitioning approach based on cutting along the cartesian plane(s). This approach can be used to ensure that an equally distributed amount of work for each MPI process is achieved and inter process communication is minimised. Referring back to Figure 2, for a structured one dimensional partition, we may simply allocate the required number of partitions by slicing the geometry by planes along the x axis. The spacing of these planes ensures that there is an even distribution of both non-boundary and boundary cells (i.e. the number of cell faces that are on a boundary, which is from 0-6 for a hexahedral cell). It's worthwhile to note that since there is a finite volume aspect to the CABARET algorithm, we have to be careful when determining neighbouring partitions. Sides alone are not enough to distinguish neighbours since partitions can be connected by a single apex. However, this situation cannot happen with a one or two dimensional data decomposition, but is possible with three dimensional and unstructured ones.

To demonstrate the performance of the MPI only CABARET for structured and unstructured partitions, the scalability of the code was measured when performing 276 time steps (without any I/O) was measured. The test case used here is a 3D backward facing step geom-

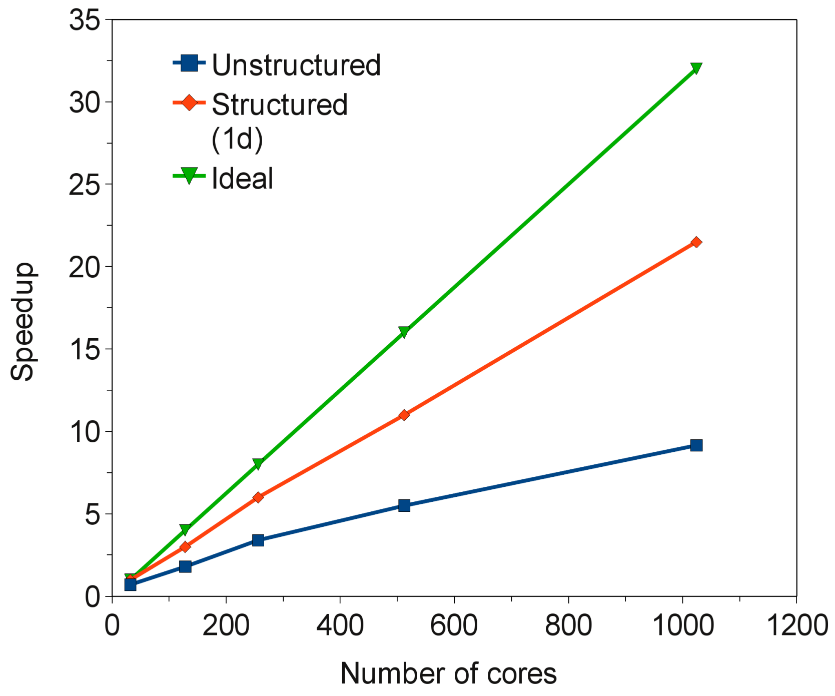

Figure 4: Scalability comparison for unstructured and structured partitioning.

etry and the boundary conditions are for laminar flow, with Reynolds number=5000 and Mach number=0.1. The grid size is 0.8×10^6 hexahedral cells. This run was performed on Phase 2a of HECToR (with each node a quad core 2.3GHz Opteron processor with 8GB of RAM) using the PGI 10.8.0 Fortran90 compiler. Figure 4 shows that there is more than a 2× speedup when using a strutured data decomposition which gives good load balancing.

4 Compiler comparison

4.1 Main CABARET Loops

In this section we shall discuss the performance of the four Fortan 90 compilers which are currently available to HECToR users, namely the Portland Group (PGI), Pathscale, Cray and GNU. The unstructured grid representation in CABARET and the associated method of indirect addressing required for the main loop structures in CABARET result in non continuously varying loop structures. These are particularly hard to achieve any vectorisation with.

4.2 Loop Structure

The following two areas of code highlight regions that would benefit from vectorisation. Each area belongs to a particular subroutine which is called every time step iteration and therefore any optimisation would be critical for improved performance of CABARET. In particular, the **PHASE2**, **BOUND** and **VISCOSITY** subroutines involve loops formed over the side of each cell. These are highlighted by CrayPat as where the most intensive computation takes place - around 60% of the cpu time:

PHASE1 and **PHASE3**

```
DO K=1,NCELL
   ...
   KEYE=KEY(K,1)
   ...
```

```
  NSI=GEMCELLSIDE(K,1)
  ...
  UI=SIDE(NSI,1)
  ....
  SXI=SIDE(NSI,11)*DFLOAT(KEYE)
  ...
END DO
```

PHASE2 BOUND and **VISCOSITY**

```
DO I=1,NSIDE
  NCF=GEMSIDECELL(I,1)
  NCB=GEMSIDECELL(I,2)
  IF((NCF/=0).AND.(NCB/=0))THEN
    CALL TAKESTENCIL1F(I)
    CALL TAKESTENCIL1B(I)
    IF (ABS(CHAR3B)<DEPS) CHAR3B=0
    IF (ABS(CHAR3F)<DEPS) CHAR3F=0
    IF(CHAR3B+CHAR3F.LE.0)THEN... ENDIF
  ENDIF
  IF(NCF<0) THEN ... ENDIF
  IF(NCB<0) THEN ... ENDIF
  ...
  NCF=GEMSIDECELL(I,1)
  NCB=GEMSIDECELL(I,2)
  NSTYPE=GEMSIDECELL(I,3)
  NC=NCF
  IF(NCF==0)NC=NCB
  NA1=GEMCELLAPEX(NC,1)
  ...
  NA7=GEMCELLAPEX(NC,7)
  IF(NSTYPE==2651)THEN
    NSA1=NA2
    NSA2=NA6
    NSA3=NA5
    NSA4=NA1
  ENDIF
  ...
END DO
```

If the compiler is able to create packed SSE instructions for these loops then some performance improvement would certainly be gained. But as the loops appear there is not much scope for vectorisation and the algorithm constraints do not allow for any code re-structuring. We have compiled the CABARET code with the default versions of the main compilers that are currently available on HECToR, i.e. PGI 10.9.0, Pathscale 3.2.99, Cray 7.2.8 and GNU 4.5.1.

The results show that none of the compilers can perform any level of vectorisation on **PHASE2**, **BOUND** and **VISCOSITY**. However, Pathscale and Cray both manage to produce partially vectorised instructions for the loops in **PHASE1** and **PHASE3**. The Cray compiler reports that it has estimated the number of vector registers required. The Pathscale compiler reports that the LOOP WAS VECTORIZED. Removing the mixed data type in **PHASE1** and **PHASE3** eliminates the type conversion from the INTEGER KEYE to REAL(KIND=8) with the intrinsic function DFLOAT and helps to achieve partially vectorised instructions for the loops with PGI.

For all three compilers, the optimisation options (i.e. compilation flags) used are to enable inter-procedural analysis, loop unrolling, loop nesting (where possible) for the target processor in 64-bit mode. In terms of producing the best performing object code for CABARET on HECToR, the Pathscale compiled version performs on average 5% faster than the PGI or the Cray generated object code. The worst performance wise is the GNU generated object code which is on average 5% slower than the PGI and the Cray generated object code.

5 Multi-core Data Parallelism

5.1 Distributed Data Parallelism in CABARET

This project began during the transistion of Phase 1 of HECToR (when each compute node was a dual core AMD Opteron 2.8GHz chip with 6GB RAM) to Phase 2a (with each chip a quad core 2.3GHz Opteron processor with 8GB of RAM). Data parallelism in CABARET for handling a partitioned Gambit unstructured mesh was initially designed for use on similar architectures, with the level of parallelisation based on MPI.

In this single program multiple data approach, the data parallelism relies upon a partitioning of the Gambit generated computational grid in order to produce sub grids, so that each one of these can be used by the individual instance of the CABARET code. Due to the unstructured decomposition used within the core CABARET algorithm an indirect referencing scheme then manages access to halo data and associated communications for the updates needed after each time step. Asynchronous MPI calls were implemented within **PHASE1**, **PHASE2** and **PHASE3** and placed so that computation could be performed while communication was in process, ensured that performance was efficient.

At the beginning of this dCSE project, the hardware setup for Phase 2b and beyond was unknown. Nearing the latter half of the work, however, it was possible to assume that this hardware would become more increasingly multi-core focussed. When Phase 2b of HECToR arrived (XT6 - 24 core Opteron 6172 2.1GHz processors, arranged as two dual hex core sockets per node in a non uniform memory architecture) it was also clear that the existing MPI parallel CABARET code would need development for multi-core architecture.

5.2 Transisition of CABARET to HECToR Phase 2b

To demonstrate the initial performance of the MPI only CABARET on Phase 2b of HECToR, the scalability of the code to perform 276 time steps without any I/O involved is shown in Figure 5. The test case used here is a 3D backward facing step geometry and the boundary conditions are for laminar flow, with Reynolds number=5000 and Mach number=0.1. The grid is 6.4×10^6 hexahedral cells. The code was compiled with the PGI 9.0.4 Fortran 90 compiler.

Figure 5 shows results for both Phase 2a (quad core) and Phase 2b (Seastar2+ interconnect) of HECToR. The interesting observation is that there is little performance degradation moving from quad core to 24 core nodes - even with the increased contention for inter-process bandwidth. In tests on both Phase 2a and Phase 2b the nodes were fully populated with the MPI tasks. In general we would expect that an increased amount of contention for interconnect use would occur due to the increased number of MPI tasks per node, but with this problem size this doesn't happen.

However, if we increase the size of the problem, very different behaviour is observed. When moving to a larger sized case of 5.12×10^7 hexahedral cells, we can see in Figure 6 that performance is clearly affected by moving from quad core to 24 core nodes. In Figure 5 the positive performance is due to the size of the problem in question being accomodated by the available cache/interconnect arrangement.

In Figure 6 we see that by running the MPI only CABARET code on Phase 2b (XT6 with Seastar2+ interconnect) using 500 and 1000 cores, the wall clock time for performing 256 iterations does decrease for all cases, whether fully populating a node (24 cores) or maintaining the same MPI process per node ratio as in Phase 2a (XT4) (4 cores). However, it is clear that the 4 cores per node case takes around half the time of the 24 cores per node case. This behaviour is expected and is a typical observation of porting a pure MPI code to a multi-core architecture.

It is worthwhile to note that running the same test on the Phase 2b XE6 (24 core Opteron 6172 2.1GHz processors, arranged as two dual hex core sockets per node in a non uniform memory architecture with a Gemini interconnect), shows an improved performance of around 17% when populating with $12 - 24$ cores per node, e.g. for 500 cores, the 24 core per node run takes 970s on Seastar2+, whereas this reduces to 805s with Gemini.

The key difference between the Phase 2b XT6 and the XE6 is the replacement of the Seastar2+ interconnect with the Gemini one. The Seastar2+ chip router has 6 links implementing a 3D-torus of processors with a point-to-point bandwidth of 9.6 GB/s. The Gemini interconnect has one third or less the latency of the SeaStar2+ interconnect and can deliver about 100 times the message throughput of the SeaStar2+ interconnect - something on the order of 2 million packets per core per second.

The Gemini interface is able to process huge numbers of small messages due to its higher injection rate in comparison to that of the SeaStar2+ interconnect. Therefore one would expect that with varying problem size, the Gemini interconnect would show considerable benefits over the SeaStar2+ interconnect for the MPI only version of CABARET.

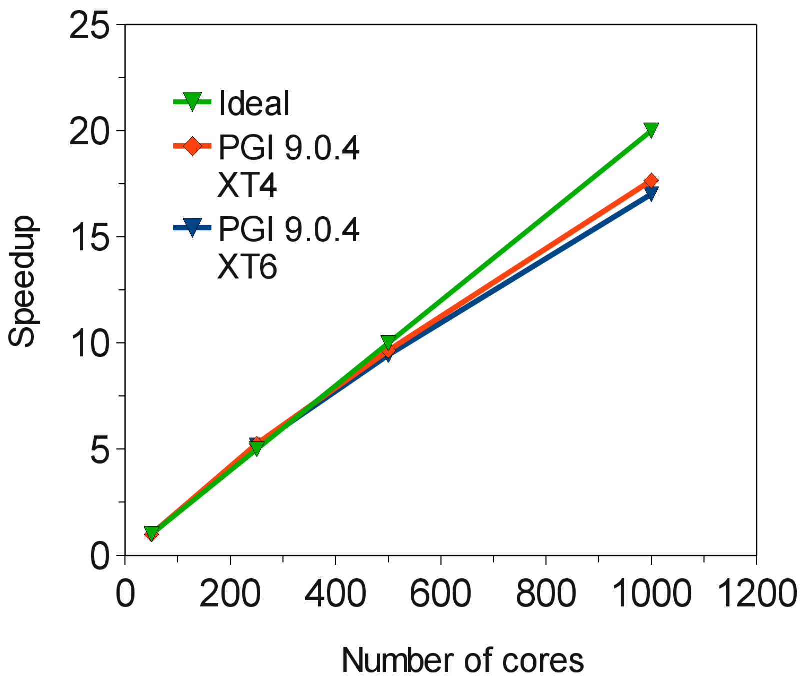

Figure 5: Scalability for a 6.4×10^6 cell backward facing step.

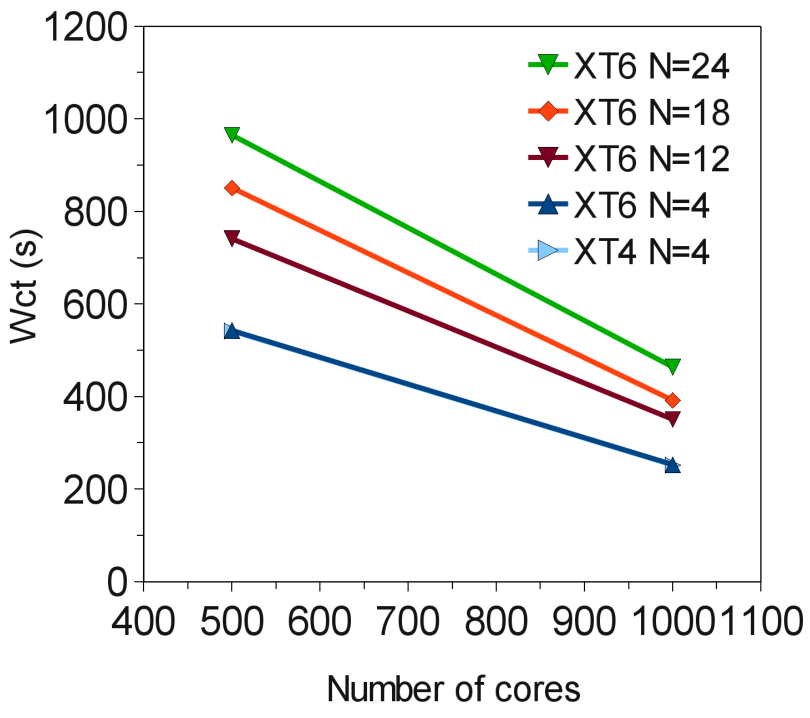

Figure 6: Time taken for 276 time steps of a 5.12×10^7 cell backward facing step.

5.3 Hybrid Parallelism in CABARET

In the pure distributed data approach to parallel CABARET, the data parallelism is carried out on partitioned sub grids of the Gambit generated computational grid. Each one of these sub grids being assigned as a single MPI process within the global communicator. Non-blocking MPI calls are implemented within **PHASE1**, **PHASE2** and **PHASE3** and placed so that computation is performed while communication is in process. The unstructured decomposition used within the core CABARET algorithm has an indirect referencing scheme which manages access to halo data and associated communications. But, to improve performance for Phase 2b we need to reduce the contention for interconnect bandwidth by reducing the number of off-node MPI communications with a shared memory approach for on node parallelism.

We have a choice when making the decision on which strategy to adopt in implementing Hybrid parallelism in CABARET.

1. Assign on node partitions to OpenMP threads rather than have them as MPI processes.
2. Parallelise the loops involving the conservative and flux variable updates, within **PHASE1**, **VISCOSITY**, **PHASE2**, **BOUND** and **PHASE3**.

Both methods will potentially reduce the number of off-node MPI communications, but 1. should effectively be already happening as part of MPT's ability to manage on node MPI processes and therefore should have little benefit, so choosing 2. makes more sense and will introduce a new level of parallelism within CABARET.

The computational stencil for CABARET involves only nearest neighbour grid points. This enables great potential for thread safe OpenMP within the main computational loops. For the loops involving the conservative and flux variable updates, within **PHASE1**, **VISCOSITY**, **PHASE2**, **BOUND** and **PHASE3**, shared memory OpenMP parallel for directives with a static schedule were implemented. These divide the loops into thread safe chunks of size NCELL/OMP_num_threads or NSIDE/OMP_num_threads and assigns a thread to a separate (local) chunk of the loop. The MPI_THREAD_FUNNELLED approach is used allowing only the master thread to perform the inter-node communication, e.g.

```
!$OMP BARRIER
!$OMP MASTER
        DO I=1,APEXNEIGHS
! pre-post required for CELL transfers

         CALL MPI_IRECV(CELLD(0,1,I),10*(MAXNCELL+1)
     &   ,MPI_DOUBLE_PRECISION,NEIGH(I),0,MPI_COMM_WORLD
     &   ,REQUESTIN(I),IERR)

        END DO ! APEXNEIGHS
!$OMP END MASTER
!$OMP BARRIER
...
!$OMP MASTER
        DO I=1,APEXNEIGHS

         CALL MPI_ISSEND(CELL(0,1),10*(MAXNCELL+1)
     &  ,MPI_DOUBLE_PRECISION,NEIGH(I),0,MPI_COMM_WORLD
     &  ,REQUESTOUT(I),IERR )

        END DO ! APEXNEIGHS
!$OMP END MASTER
...
       DO I=1,APEXNEIGHS
        CALL MPI_WAIT(REQUESTOUT(I),STATUS,IERR)
       END DO ! APEXNEIGHS
```

All multi-threaded loops are able to run with NOWAIT, again due to the need to consider only nearest neighbour grid points.

It is also more useful to implement OpenMP within the **PHASE1**, **VISCOSITY**, **PHASE2**, **BOUND** and **PHASE3** loops, since these take around 60% of the cpu time and will not vectorise for the reasons discussed in Section 4.2.

5.4 Performance of the Hybrid Parallel CABARET code

In this section we shall discuss the performance of the hybrid parallel CABARET. We shall revisit the 5.12×10^7 cell backward facing step, again with boundary conditions for laminar flow, Reynolds number=5000 and Mach number=0.1. For each test run we can vary the number MPI processes (n), the number MPI processes per node (N) and the number of OpenMP threads per MPI process (d). There is always one MPI process per hex core die or per socket.

Figure 7 shows the speedup for 50, 250, 500 and 1000 MPI processes for 276 time steps. A comparison can be made between the two runs, which are shown for:

1. 4 MPI processes per node with 6 OpenMP threads per MPI process.
2. 2 MPI processes per node with 12 OpenMP threads per MPI process.

The two runs were carried out on HECToR Phase 2b with the Seastar2+ interconnect. Case 1. performs slightly better than Case 2. This is a demonstrative case and for a variation in problem size, the situation may be slightly different. The ratio between computation and

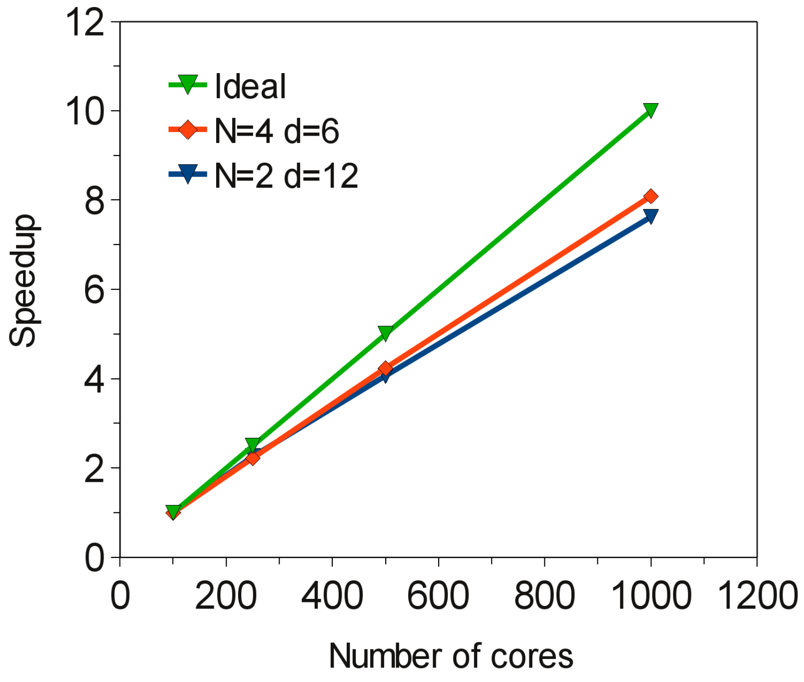

Figure 7: Scalability for a 5.12×10^7 cell backward facing step.

communication, is an important factor in CABARET performance since the main computational loops are non vectorised. So for any particular problem size, it would be recommended to test which setup gives best performance for the number MPI processes per node and the number of OpenMP threads per MPI process for a given number MPI processes (partitions). It's worthwhile to note that performing the same simulation with the Gemini interconnect does give a $5 - 10\%$ speedup, which is expected since this computation is using a one dimensional data decomposition and is not heavily dependent upon the performance of communication.

6 CABARET Demonstration

To demonstrate the method, Figure 8 shows the normalised x direction velocity component of the flow field at 40000 time steps the evolving solution after 10000 time steps for the 3D backward facing step test case. Here, the boundary conditions are for laminar flow, with Reynolds number=5000 and Mach number=0.1. The grid size is 10 points per the step height giving a total grid size of 800000 points. Time-averaged velocity fields shown for this mesh resolution and for a 20 cells per step height give the correct length for the recirculation zone around the step as shown in Figure 9. The Vorticity (Q-norm) iso-surfaces for the grid density of 20 cells per step are shown in Figure 10.

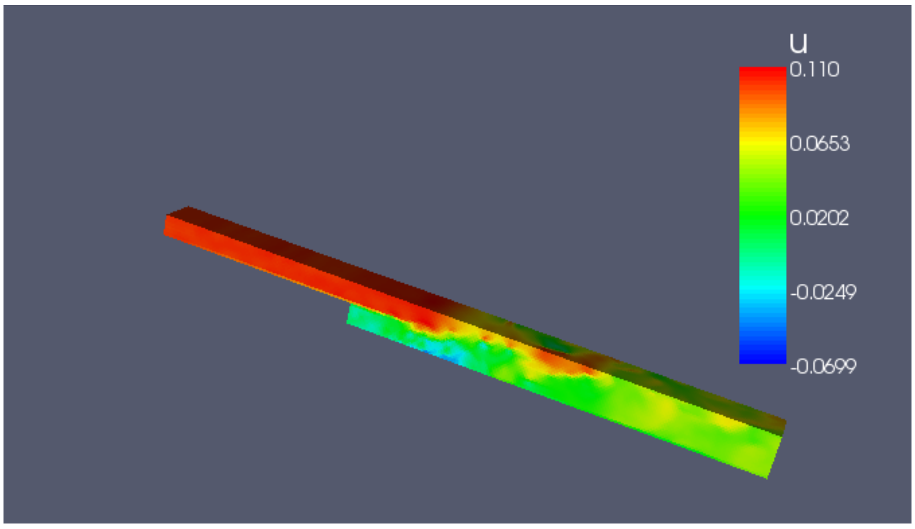

Figure 8: x direction velocity component of the flow field at 40000 time steps

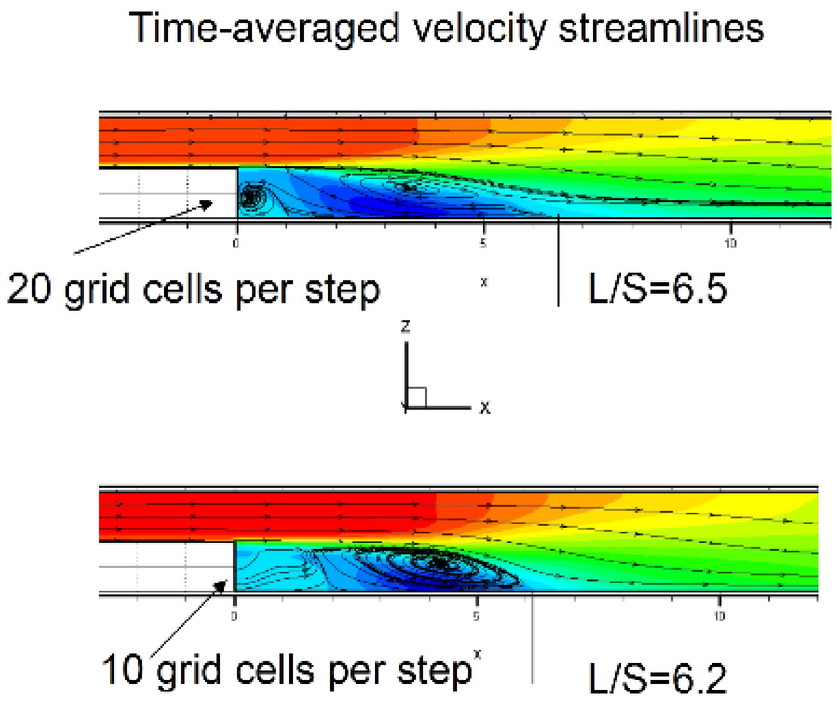

Figure 9: Time-averaged velocity fields for two mesh resolutions

7 Conclusion

Through a dedicated Computational Science and Engineering project sponsored by the HECToR dCSE programme, the general purpose CFD application CABARET has been ported to run efficiently on the HECToR UK national supercomputer resource. Scalability of the CABARET application has been demonstrated. This work will now enable leading research to be performed on MILES modelling and general turbulence simulations and introduce a step change in the capability of the application.

In this report, we have described the development of a scalable hybrid parallel implementa-

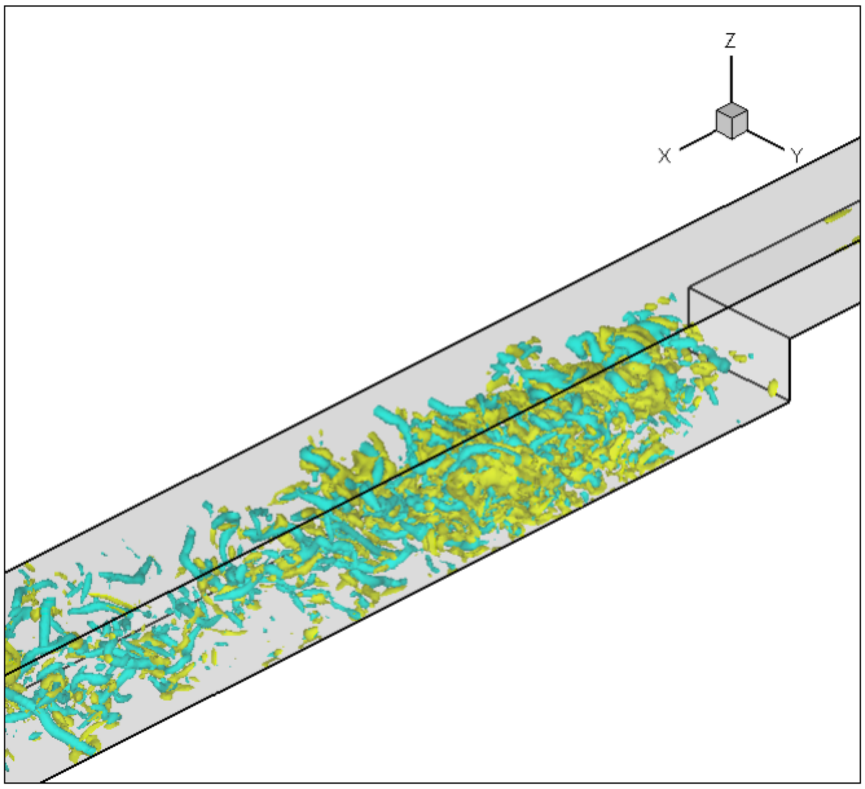

Figure 10: Vorticity (Q-norm) iso-surfaces for the grid density of 20 cells per step

tion of the CABARET application. This has taken place over the Phase 1 to Phase 2b (with Gemini) lifetime of HECToR. The CABARET method is particularly suitable for modelling high Reynolds number low Mach number turbulent flows in Large Eddy simulations. The method ensures that only nearest neighbour cells are involved and is therefore suited to distributed computing architectures.

The input grid is a Gambit(Fluent) produced unstructured hexahedral mesh and we have implemented an automated method of grid partitioning - using either a structured (cartesian plane) decomposition or unstructured external package - METIS/Scotch. Where the geometrical arrangement permits, users are recommended to employ a structured partitioning to ensure efficient load balancing. An MPI parallel version of CABARET was developed and tested on Phase 2a. The format of the main computational loops in CABARET make it difficult to achieve any form of vectorisation with any of the Fortran90 compilers available on HECToR.

The arrival of Phase 2b and increasingly more multi-core focussed architecture brought the need for a hybrid parallel version of CABARET. For a representative size test case, it was shown that the performance of a pure MPI version of CABARET was not as good on the 24 core per node Phase 2b architecture as it had previously shown on the quad core nodes of Phase 2a. Implementing hybrid parallelism in the form of multi-threaded work sharing for the main loops has helped performance and enabled more efficiency of multi-core use of CABARET.

The overall outcome of this work may be summarised as follows:

1. An automated geometrical domain decomposition method for partitioning a Gambit generated unstructured grid. The resulting partitions are ensured to give good load balancing for the main computation and minimise communication. The partitioner was demonstrated on a 5.12×10^7 grid and is currently capable of processing a 2.05×10^8 grid on HECToR

Phase 2b. An internal report summarising methods for unstructured partitioning was written [13].

2. A parallel version of CABARET was implemented using non-blocking MPI to pass data between internal boundaries (cell faces and sides). This code was validated and tested against the old code using the 3D backward-facing step case and a parallel efficiency of 72% was observed when using 250 cores of the XT4 part of HECToR with the quad core Phase 2a architecture.
3. The new code was validated for a larger test case of 5.12×10^7 grid points on the XT4 part of HECToR, using the 3D backward-facing step case.

In addition to meeting the original aims of the project, the follow has also been achieved:

4. Implementation of a hybrid (OpenMP/MPI) parallel version of CABARET for Phase 2b and further. Also a hybrid parallel partitioner has been developed for the domain decomposition of the Gambit generated unstructured grid.
5. A 3D backward-facing step test case of 5.12×10^7 cells, retains 80% parallel efficiency on 1000 cores of Phase 2b.
6. Simulations can now be performed on grids at least 512$\times$ larger than previously possible. Smaller scale simulations can now be performed in a few minutes that would otherwise have taken several days before this work.

The main users of CABARET are now able to investigate problem sizes at least 100 times larger than previously possible. Particular HECToR use of CABARET includes 7M AUs which was awarded at the November 2010 RAP for the project entitled "Turbulent Flow Effects on Flap Noise". A further HECToR resource of 21M AUs over Phase 2b and over 21M AUs for Phase 3, for further investigation in high Reynolds number turbulent flows of Jet Flap noise (EP/I017747/1) will use CABARET as the main code. CABARET is available to all HECToR users by contacting the HECToR CSE team (support@hector.ac.uk).

8 Acknowledgments

This project was funded under the HECToR Distributed Computational Science and Engineering (CSE) Service operated by NAG Ltd. HECToR - A Research Councils UK High End Computing Service - is the UK's national supercomputing service, managed by EPSRC on behalf of the participating Research Councils. Its mission is to support capability science and engineering in UK academia. The HECToR supercomputers are managed by UoE HPCx Ltd and the CSE Support Service is provided by NAG Ltd. http://www.hector.ac.uk

References

[1] "Balanced Characteristic Method for Systems of Hyperbolic Conservation Laws", V.M. Goloviznin, Doklady Math., 72 (2005), pp619-623.

[2] "A New Efficient High-Resolution Method for Nonlinear Problems in Aeroacoustics", S.A. Karabasov and V.M. Goloviznin, American Institute of Aeronautics and Astronautics Journal Vol. 45 (2007), pp2861-2871

[3] "A novel computational method for modelling stochastic advection in heterogeneous media", V.M. Goloviznin, V.N. Semenov, I.A Korortkin, and S.A. Karabasov, Transport in Porous Media, Springer Netherlands, 66(3) (2009), pp439-456.

[4] "Compact Accurately Boundary Adjusting high-REsolution Technique for Fluid Dynamics", S.A. Karabasov and V.M. Goloviznin, Journal of Computational Physics, 228 (2009), pp74267451.

[5] "CABARET in the Ocean Gyres", S.A. Karabasov, P.S. Berloff and V.M. Goloviznin, Journal of Ocean Modelling, 30 (2009), p55168.

[6] "Generalized Leapfrog Methods", A. Iserles, IMA Journal of Numerical Analysis, 6 (1986), No 3, pp 381-392.

[7] http://www-users.cs.umn.edu/∼karypis/metis/

[8] http://www.labri.fr/Perso/∼pelegrin/scotch/

[9] "A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs", G. Karypis and V. Kumar, SIAM Journal on Scientific Computing Vol. 20 (1999), pp359-392.

[10] http://www.cecill.info/licenses.en.html

[11] "Static mapping by dual recursive bipartitioning of process and architecture graphs", F. Pelegrini, Proceedings of the IEEE Scalable High-Performance Computing Conference (1994), pp486-493.

[12] "A multilevel algorithm for partitioning graphs", B. Hendrickson and R. Leland, Proceedings of the IEEE/ACM SC95 Conference(1995), pp28-28. Also available at http://www.sandia.gov/ bahendr/papers/multilevel.ps

[13] http://www.hector.ac.uk/cse/reports/unstructured_partitioning.pdf