Direct Numerical Simulation (DNS) of turbulent fluid flows - a dCSE Project

Ning Li
Numerical Algorithms Group (NAG)
Wilkinson House, Jordan Hill Road,
Oxford, OX2 8DR, UK

April 29, 2012

Contents

1 Introduction

This project concerns the further development of Incompact3D, an incompressible Navier-Stokes solver, and its sister codes, Compact3D & QuasiCompact3d, for compressible flow simulations. These applications are used by the Turbulence, Mixing and Flow Control group at Imperial College and its academic collaborators to conduct state-of-the-art turbulence studies.

This is the second dCSE project for code Incompact3D. In a very successful first dCSE, the scalability of Incompact3D was significantly improved via the creation of 2DECOMP&FFT, a 2D pencil decomposition communication framework with a distributed FFT interface [3]. As a result, Incompact3D now regularly uses 4,000-16,000 cores on HECToR for production runs.

The current project contains two major work modules. First, the two sister codes of Incompact3D, both in serial form originally, are to be parallelised using the same 2DECOMP&FFT framework. This is to enable the research group to simulate large-scale compressible turbulent flows. Second, a new communication library is to be introduced to enable embedding explicit numerical schemes within the framework of an spatially implicit node. This makes it possible to use Incompact3D to study a wider range of problems, such as particle-turbulence interaction.

The project was granted 6 months full-time effort for one person to work on a full-time basis, which translates to 1 year effort on a 50% part-time basis. The project officially started in March 2011.

2 General serial code modernisation and improvement

The Incompact3D family of applications have a long history. Unlike the Incompact3D code, which had already been modernised during the previous dCSE project, the two sister codes were still in old Fortran 77 style and required major updates before the parallelisation. Following is a list of key works done during this project:

- All source codes have been converted from fixed-format to free-format Fortran.
- All common blocks for data sharing have been replaced by Fortran modules.
- All major data structures have been converted from static to allocatable arrays.
- Most arrays containing important flow variables have been defined in a module shared by major functional units, significantly reducing the number of parameters passing between subroutines and improving the readability and maintainability of the codes.
- A lot of cosmetic changes have been applied, including passing the source code through NAG compiler's polisher (an option in the latest NAG compiler version 5.3; a standalone utility in early versions), in order to follow a set of established coding styles and best practices.
- Wherever possible, subroutines newly rewritten for Incompact3D have been adapted and reused in the sister codes. Although incompressible and compressible Navier-Stokes solvers are fundamentally different, some key algorithms, such as spatial derivative computations, may be shared. Such practice also helped to further validate these routines. Compact3D contained a few bugs in its derivative calculations. These have been discovered by cross-checking with the related Incompact3D routines. At the end of this project, about 2500 lines of source code are shared across the Incompact3D family of codes.
- Some obsolete code features, such as swapping the dimensions of 3D arrays, which was historically used to optimise code performance on old vector machines (to increase vector length), have been removed to simplify future code development.

- Another outdated code feature has also been identified: the arrays to store the stress tensors (used when evaluating the convection and diffusion terms of the Navier-Stokes equations) are used as temporary work-space elsewhere. This was to reduce the memory footprint of the codes on old systems but this makes the code very hard to understand and maintain. In parallel implementations, memory constraint is normally no longer an important factor. To solve this problem, the code author would like to introduce a new set of properly named variables after consulting key users. So this improvement was not implemented during this project.

The original serial codes were imported into an SVN server for version control. This is the same SVN server used by the previous Incompact3D project. Code validations based on a standard test case were carried out after each step change. After making all the changes, the new serial codes, although look beyond recognition, can still produce identical results as the original.

These code modernisation works were not originally planned in the proposal but was found to be necessary to ensure the high-quality parallelisation works later.

3 Strategy for parallelisation

As proposed, the code parallelisation makes use of the 2DECOMP&FFT library created in the previous dCSE project.

Because of the flexible design of the 2DECOMP&FFT library, there are a number of ways to parallelise the serial Compact3D and QuasiCompact3D codes. After consulting the code author and key users, the following strategies were used:

- All screen outputs are handled by MPI rank 0 only.
- All 2D/3D variables are distributed using the 2DECOMP&FFT framework.
- 1D arrays remain 'global' arrays - i.e. each MPI process maintains a copy of them, even though some information not 'local' to a process may never be used. This only marginally increase the memory usage but simplify the coding.
- The 2D/3D distributed arrays always use global coordinate. This simplifies a number of operations in the codes without causing any inconvenience. Several 2DECOMP&FFT's communication and I/O routines have since been updated to handle data defined in both local and global coordinates.
- All distributed arrays are initially defined in X-pencil format, therefore having a one-to-one mapping to the variables in the original serial code. They are only transposed to other decomposition if required.
- For a few primary flow variables, such as velocity, density, pressure and temperature, three copies of same variable are stored, each corresponding to one pencil orientation as defined in 2D pencil decomposition. This simplifies coding but may introduce additional communication overhead, which will be further discussed later.
- For global operations, such as finding global maximum/minimum or averaging along global mesh lines, MPI reduction is used. Because these are very common tasks in other applications as well, a library routine is planned in the next release of the 2DECOMP&FFT library to facilitate this.
- For transpose-based parallel codes, original serial algorithms often need to be adjusted (for example, by reordering computational steps, using temporary work-spaces for intermediate

results, and duplicating computations) in order to minimize the number of communication operations. The code authors prefer to perform the above-mentioned optimisations by themselves so that they have full control over the key algorithms in the parallel code. So during this project, the codes were parallelised in such as way that one-to-one mappings can be found between small sections of the parallel algorithms and the original serial algorithms. This will affect the absolute performance (we may have performed more transposes than actually required) but not the scalability of the parallel codes.

This set of procedure was designed to help future code development. Should the developers need to add new functionality, they can plan their new algorithms as if they are writing a serial code. By following the strategies documented above, new algorithms can be easily parallelised to the same standard.

Validating an MPI code is normally only possible when all development works are completed. During this project, a procedure was established to test the intermediate codes regularly by running them on a single MPI rank (with all communication code being covered and tested on a local node), which should always produce identical results as the serial code. This helped pick up a number of coding errors.

When the parallelisation was completed, the code was moved onto a cluster for debugging. The main procedure was to compare intermediate results stored in distributed array to those stored in the corresponding arrays in the serial code. Code was checked section by section until all data matched to machine accuracy. Further validation was done by evaluating integral quantities (such as system entropy) over the computational domain - they should remain constants regardless of the domain decomposition approaches. In the end, the parallel code was shown to produce identical numerical results as the original serial code.

4 Parallel performance

The new code was ported onto HECToR for further validation and benchmark. The performance study on HECToR was based on a test case supplied by a key Compact3D user. This is a small simulation using a 3D mesh of $75 \times 101 \times 75$ points. Note that this problem size is about 2 magnitude smaller than the typical large production simulations. Due to the time constraint of this project, and the complexity setting up large-scale simulation, it was not possible to benchmark a more realistic case.

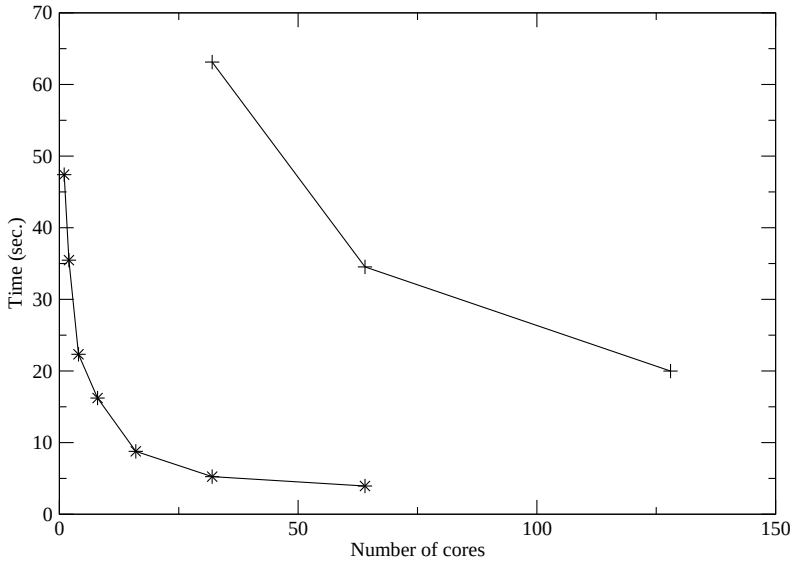

Figure 1: Strong scaling of Compact3D.

Figure 1 shows the strong scaling of Compact3D. First of all, it can be seen that the absolute speed-up over the serial run is not very impressive - 12 times faster when using 64 cores. However, such performance is broadly in line with the behaviour of the base 2DECOMP library - the ALLTOALL type of communication introduces significant overhead initially at smaller scale, but good scaling at large parallel runs can be observed. Such claim can be supported by a second set of benchmark in which the number of mesh in the small test case was artificially doubled in each dimension. The code demonstrates much improved scaling with a parallel efficiency of nearly 80% between 32 and 128 cores.

5 Implementation of a new communication library

Incompact3D is parallelised using transpose-base method because its key algorithms are spatially implicit and requires boundary-to-boundary data access. There are, however, many situations in which stencil-based explicit schemes are desirable. For example:

- In large-eddy simulation (LES), the small-scale turbulence is to be modeled using the resolved flow field. To construct the subgrid-scale model, spatial derivatives often need to be evaluated. Using implicit formulations for such derivative calculations would be expensive and unnecessary (one does not normally require any order of numerical accuracy for modelled quantities).
- Particle tracking[1], used in applications such as particle-turbulence interaction, is easier to be implemented with a fixed domain decomposition and explicit numerical schemes.

To support explicit numerical schemes, a second set of communication routines have been introduced. This involves building halo cells around the pencils (as defined in the 2D decomposition) and writing point-to-point communication code to enable neighbouring pencils to exchange data.

The original target, as outlined in the proposal, was to perform particle tracking only in X-pencils and using a 4th-order finite difference scheme for spatial interpolations (with five-point stencil, therefore requiring two layers of halo cells while building the communication code). However, because of the usefulness of these explicit schemes, a more general communication routine has been designed and implemented in 2DECOMP&FFT to support any number of halo cells.

In a related work, the P3DFFT package[2] (also use a 2D pencil decomposition) implemented its halo-cell support using the following storage arrangement:

```
|000000000000000000000000000|11111|22222|33333|44444|55555|66666|
             ^                 ^     ^     ^     ^     ^     ^
        Pencil data            W     E     S     N     B     T
```

As shown, the first section of the memory contains the data in a pencil. The data in halo cells is then appended to the pencil data, in the order of west/east (the 1st dimension), south/north (the 2nd dimension), and bottom/top (the 3rd dimension). The advantage of such storage format is its efficiency - the same memory reference can be used in the global transposition routines, where the halo cells are simply ignored. The disadvantage is that extra code has to be written in applications to map the halo-cell data to a more convenient (i,j,k)-indexed format before being used.

In the current project, a more user-friendly design was adopted. A single library routine has been introduced in 2DECOMP&FFT, in the form of:

[1]Note that partitioning computational domain for particle tracking, as discussed in this project, is not suitable for large-scale problems. When particle tracking is computationally more expensive than obtaining the base flow field, if the particle distribution is not even, then domain decomposition can introduce significant load-imbalance. Consider using a task-based decomposition instead if that is the case.

```
call update_halo(var, var_halo, level)
```

Here the first parameter *var*, an input array, contains the normal pencil-distributed data as defined by the domain decomposition. After the subroutine call, the second parameter *var_halo*, an output, returns all original data plus halo data collected from the neighbouring processes through MPI. The third parameter *level* simply defines how many layers of overlapping is required. *var_halo* should be defined from the calling routine as either a 3D allocatable array or pointer. Its memory space will be calculated and allocated by the library and when the subroutine returns it can be indexed by the calling program using the normal i,j,k indices. A more general form of this subroutine is also available:

```
call update_halo(var, var_halo, level, opt_decomp, opt_global)
```

The two optional parameters, *opt_decomp* and *opt_global*, enable the use of arbitrary global data sizes and global coordinates in data storage.

For transpose-based algorithms, boundary conditions can normally be applied to the dimension completely residing in local memory as they are done in serial code. The library code does provide direct support to periodic conditions so that pencils bordering one side of the domain may exchange information with those in the other side.

The library internally handles the communication between neighbouring pencils using standard MPI non-blocking point-to-point communication routines. Full details of the halo-cell support is available in the 2DECOMP&FFT documentation [1].

The validation of the new halo-cell communication was straight-forward. This is because any communication that can be achieved using halo-cell exchanges can also be performed using global transpositions, albeit much slower. The two sets of communication code validate each other.

The new communication code is now being used by Incompact3D to implement several key algorithms. For example, when using the Immersed Boundary method to model solid bodies in the computational domain, Incompact3D needs to perform 3D interpolations to prescribe the correct boundary conditions on the fluid-solid interface. Such interpolations, using explicit numerical schemes, are made possible by the new communication code. Its particle tracking algorithm, which involves interpolating velocity field from mesh points to arbitrary spatial locations, also requires the halo-cell support.

6 Concluding Remarks

Through this dedicated software engineering project, Incompact3D's sister codes have been parallelised. A second communication framework has been introduced to expand Incompact3D's capabilities. All major objectives in the original proposal were achieved.

As discussed in Section 3, there is still significant scope for improving the performance of the new parallel code, by adjusting its algorithm to make better use of the communication library. Such work is more of an algorithm design nature than software engineering. The PI and his associates are provided a good platform to implement further improvements.

Like in the previous dCSE project, reusable software components have been extracted and put in the 2DECOMP&FFT library. A new version 1.4 of 2DECOMP&FFT was released in October 2011.

7 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] 2DECOMP&FFT website. http://www.2decomp.org.

[2] P3DFFT website. http://code.google.com/p/p3dfft/.

[3] N. Li and S. Laizet. 2DECOMP&FFT - a highly scalable 2D decomposition library and FFT interface. In *Cray User Group 2010 conference*, May 2010.