Distributed CSE Support for HECToR-enabled Step Change in Turbulent Multiphase Combustion Simulation

Lucian Anton
Numerical Algorithms Group Ltd,
Wilkinson House, Jordan Hill Road,
Oxford, OX2 8DR, UK

September 14, 2011

Abstract

DSTAR simulates multiphase reactive flows in which complex interactions exist among vortex dynamics, entrainment, mixing, turbulence, combustion and evaporating droplets. The macroscopic turbulent behaviour and thermodynamical properties are obtained using direct numerical simulation from the original governing equations. The work done in this dCSE project will enable DSTAR to simulate new physical regimes over larger domains and longer times scales by increasing parallel scalability, serial performance and efficient input/output operations for large data sets.

Contents

1 Introduction

1.1 Project summary

The following work packages and objectives were proposed for this six month dCSE project with the aim to increase DSTAR scalability and usability:

WP1: Implementation of a 2D domain decomposition algorithm

(a) To convert the base data structures for the new domain decomposition.

(b) To update the core fluid solver and run a basic validation test.

(c) To update the remaining multi-physics modules.

(d) To benchmark the new DSTAR code.

Objective: DSTAR should scale to at least 10,000 cores with 80% scaling efficiency.

WP2: Improved I/O for capability size data sets

(a) To refactor the checkpoint procedures and statistics procedures for handling large data sets

(b) using MPI-IO type of parallel I/O solution.

(c) To re-engineer the logging procedures.

(d) To port some of the common I/O routines back to the 2DECOMP&FFT library. But only if this is worthwhile.

Objective: DSTAR should be able to efficiently handle datasets for cubic grids of at least size 500.

WP3: Code refactoring. Most of this will be done along with the code update in WP1. The remaining routines, in particular 'user routines' will be modernised using Fortran 95 as these are regularly updated by scientists who use DSTAR. Some legacy library routines may be kept in F77 form unless any significant performance issue is identified.
Objective: All user routines of DSTAR should be modernised.

WP4: Dissemination
Objective: All user routines of DSTAR should be modernised and a brief user document provided for future DSTAR users.

1.2 Work summary

From the perspective of simulating new physical regimes with DSTAR the main task of this project was to implement a two dimensional domain decomposition with the help of the 2DECOMP library described in Ref [2]. The bulk of the work for this task was to adapt the local arrays which are used to store the local grid data, and a number of subroutines to the 2D domain decomposition. More details are presented in Sections 3, 5.
The new code was then tested with up to 18,432 MPI tasks and shows good scaling efficiency (approximately 50%), although the 80% efficiency target in WP1 was not reached. The original target was set from using scalability data for other fluid dynamics codes. However, when this target was set, the fact that DSTAR has a multicomponent nature was overlooked. The multicomponent aspect results in a very large amount of data communication in the two dimensional decomposition, hence the loss in efficiency with respect to other fluid dynamics models.
We have also pursued this matter further because at large MPI task counts the code spends approximately 50% of the run time doing communication. We have found that a significantly better scalability can be achieved with an algorithm that overlaps communication with computation using a Mixed Mode programming model (see Section 3, Ref [3]) but this is at the cost of a rather more complex source code.
The input/output (IO) operations were changed from a multiple file access to a single file access pattern. This replaces the original output mechanism which used a large number of small files on the HECToR parallel file system and was therefore very inefficient. In the new version, the restart file (of binary data) can be accessed with MPI-IO although we have found that good tuning of Fortran binary IO is also efficient as well, see Section 4. In addition to restart data, there is also separate IO for monitoring points regarding physical quantities.

This data is now collected in one ASCII text file and handled by a designated MPI process. An auxiliary program has been written to sort out the data from the monitoring points into individual files for the required visualisation. The log mechanism was updated to the same IO model. Details of this work are presented in Sections 4, 5.
Code modernisation was pursued in two main directions: i) replacing static arrays with allocatable ones, this allows one to use larger grids sizes and to use one executable for arbitrary grid sizes; ii) group Fortran 77 subroutines in modules according to their functionality. This transformation has made the source code structure clearer. It also helps error location and offers the possibility to write more simple code for the top level subroutines. For better control over data structures implicit data types were replaced with explicit declarations within most of the source files. Optimisation work was also done on some subroutines for the fluid solver which brought approximately a 20% speedup for the core solver, more details are presented in Section 5.

2 Background and pre-dCSE code overview

For the last 20 years, DSTAR has been used, in original and modified versions, by a number of researchers who have worked closely with Prof. Kai Luo on turbulence and combustion simulations with the direct numerical simulation method. These include: Prof. Neil Sandham and Dr. Zhiwei Hu (University of Southampton), Prof. Xi Jiang (University of Lancaster), Dr. Eldad Avital (Queen Mary, University of London), Dr. Yongman Chung (University of Warwick) and Dr. Jun Xia (Brunel University).
The system of partial differential equations that sets the mathematical frame for DSTAR and the numerical algorithms for their solution are described in detail in Ref [1]. In short, implicit compact finite difference schemes are employed for spatial discretisation with sixth-, fourth- and third-order schemes for internal, near-boundary and boundary points, respectively. A fourth-order Lagrangian interpolation scheme is utilised to obtain gas properties at droplet locations. A third-order explicit Runge–Kutta scheme is used for temporal advancement of the flow variables, while a semi-analytical scheme is employed for droplet marching, with the consideration of numerical accuracy, stability and efficiency.
In the pre-dCSE version of DSTAR the parallel computation utilised a primary Cartesian grid, which was partitioned uniformly over the MPI ranks along y axis (1D decomposition). Because the spatial derivatives are computed with an implicit rule, each MPI rank also needs a local

grid which spans the whole domain along the y axis, hence a secondary partition along z axis is also needed. Data between the two partitions are exchanged by transpose operations among all the MPI ranks, see Figure1 a).
The numerically intensive work which is done in DSTAR takes more than 90% of the run time for a typical run. This is concentrated within the subroutine which computes the right hand side terms (RHS) for the set of differential equations in time. These equations describe the evolution of the flow variables after spatial discretisation of the continuum field equations. The RHS set of subroutines contains mainly loops that perform updates to the flow variables, together with some derived quantities inside the domain and at the boundaries. Additionally, calls to subroutines that compute the spatial derivatives and calls to communication subroutines for the transpose operations described earlier are also included.
This section of the code has a large degree of parallelism due to locality of the update rule and the multicomponent nature of the discretised flow variables. On the other hand, for a global grid with dimensions N_x, N_z, N_y the number of processors that can be used in 1D domain partition is limited by $min(N_y, N_z)$, which could be significantly smaller than the number of processing elements available on today's top supercomputers.
The input, output operations for the monitoring and the checkpoint data files were done with Fortran IO in multiple files, the monitoring data was collected in (ASCII) text files, one per observation point, the restart data was collected in binary files associated with subgroups of MPI tasks.
The source code files included a significant proportion of Fortran 77 files, many of them using implicit data declarations. The grid dimension and the number of MPI ranks were hard wired in an include file, thus making it necessary to recompile the source code if any of these parameters was changed.

3 2D domain decomposition

Fig 1 shows the schematic difference between a one dimensional decomposition and a two dimensional decomposition of the global grid. The 2D decomposition increases the number of MPI tasks with a factor equal to the number of rows used, but because the spatial derivatives are computed with an implicit rule, one more transpose operation along the x axis is needed in the case of a 2D decomposition. Nevertheless for large grids and a high speed communications network, this procedure is scalable.

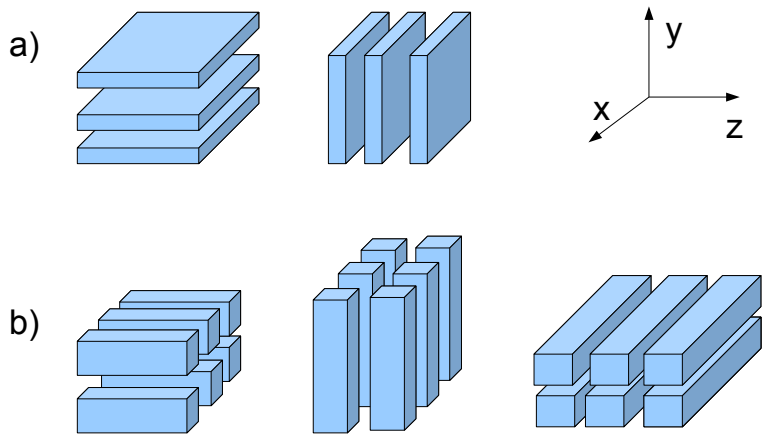

Figure 1: Schematic representation of the domain decompositions required by DSTAR: a) 1D case with 3 MPI ranks along the z and y directions, b) 2D case with 3×2 MPI ranks along the z, y and x directions; in this case a transpose along the x axis is also needed. Data exchange between decompositions require all to all type of communication.

The implementation of the 2D domain decomposition is straightforward with the help of the 2DECOMP library [2]. The calls to subroutines used for the $y \leftrightarrow z$ transposed provided in the original code were replaced with calls to the corresponding `transpose_y_z` subroutines of the 2DECOMP library. In the case of the x derivative, which needs a $y \leftrightarrow x$ transpose the subroutine that computes the derivative was extended to perform the transpose operation if needed (this is determined from the analysis of the local grid dimensions). The possibility to use a 1D decomposition is preserved, in this case the code skips any $y \leftrightarrow x$ transpose operation thus preserving the performance of the initial version.
The scalability of the 2D decomposition algorithm was tested on two cubic grids with linear sizes 768 and 1536 using up to 18432 MPI tasks. The number of processor rows used in the processor grid was set to 24 or 48 in order to keep the data movement for the $y \leftrightarrow x$ transpose inside a node or only between two nodes.
Figure 2 shows the computation time per grid point (normalised to the size of 768^3 grid) of the core solver subroutine (RHS) which in for a typical production run will consume more than 90% of the computation time. The plot shows that the 2D decomposition has a good scaling up to 18432 MPI tasks with approximately a 50% efficiency for

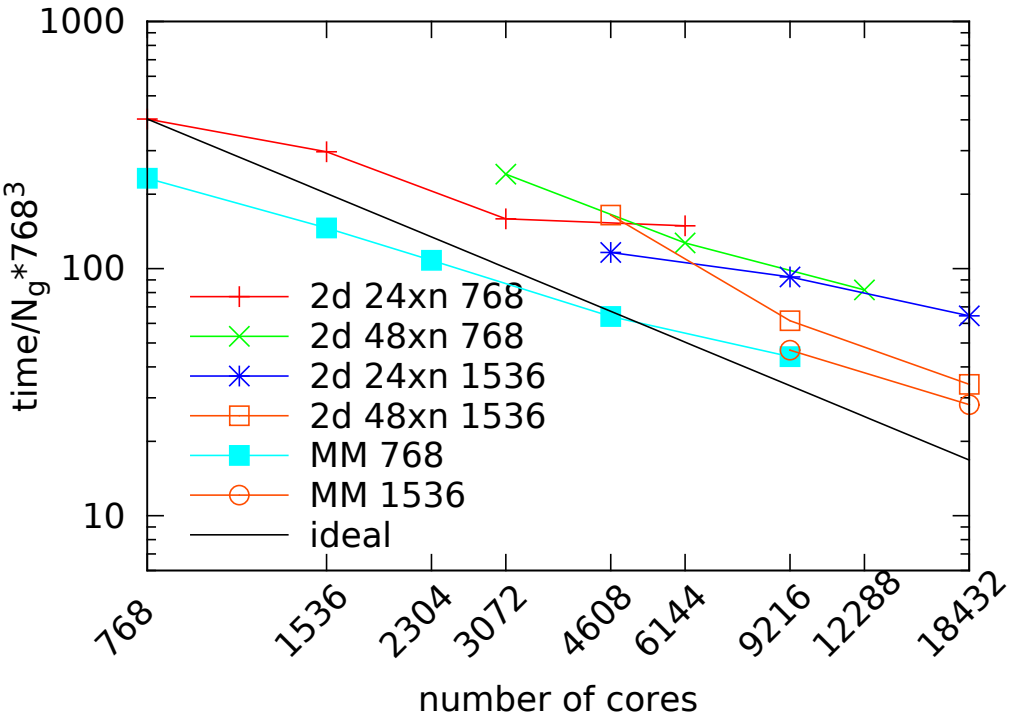

Figure 2: Performance of the core solver subroutine (RHS) versus total number of MPI ranks in the case 2D decomposition with 24 and 48 processor rows for two cubic grids with linear size 768 and 1536. Note the crossover for both grids as the total number of ranks increases. It is also shown that the performance of the mixed mode code (MM) which uses a fixed 1D decomposition with 768 and 1536 MPI tasks for each grid, respectively, but with increasing number of OpenMP treads.

grid size	MPI tasks	code sections					
		RHS		monitor		restart	
		old	new	old	new	old	new
$232 \times 384 \times 768$	192	13.3	10.8	0.22	0.12	4.3	23.1
768^3	768	31.1	25.0	0.51	0.01	21.1	34.7
1536^3	18432	-	31.1	-	0.14	71.1	226.6

Table 1: Timings for the pre-dCSE version and the current one for three grid sizes. Data are for the right hand side term computation (RHS) (seconds per step), monitoring (seconds per operation) and one restart (read and write). The first two models use a 1D decomposition, the last one uses a 2D decomposition with 48 processor rows. The pre-dCSE code cannot run the largest grid because of insufficient memory, the restart time was obtained on the current version with the Fortran IO switched on. The restart files for the MPI-IO version use 1, 32 and 64 stripes, respectively.

the grid size 1536 using 48 rows for the MPI task grid. An interesting crossover phenomenon is worth mentioning here. The performance of the 2D decomposition with 24 processor rows is better at small core counts: (3072 MPI tasks for the 768^3 grid and 4608 MPI tasks for 1536^3 grid) but the decomposition that uses 48 processor rows has better performance at larger numbers of MPI tasks for both grid sizes (other tested values for the number of rows show a significantly lower performance).

It has not escaped our notice that on a parallel computer with largely multicore nodes, an alternative algorithm can be used to increase scalability: one can keep the original 1D decomposition of the simulation grid, thus saving a significant amount of communication, and increase the speed of computation with the help of OpenMP threads in each local grid.

A prototype implementation of this algorithm was implemented, together with a thorough study of scalability. This work was presented at the Cray user Group conference 2011 [3]. The main conclusion of this study is that a mixed mode 1D decomposition is at least 20% faster than a 2D decomposition at a cost of more complicated code, see Fig 2, even better results area achievable if overlapping between communication and computation is done with the help of OpenMP threads[1].

[1]This is required because the current MPI implementations do not provide computation-communication overlap for non-blocking send receive operations

4 Optimisation of Input/Output operations

DSTAR uses IO for two main tasks: i) collection of physical quantities from a selected set of monitoring points inside the simulation domain and ii) checkpoint data needed to restart the computation.
In the pre-dCSE version of the code, the monitoring data was collected in ASCII text files with one per observation point. This could lead to hundreds or thousands of individual files being access over the parallel file system.
The restart data was written in binary format by a subgroup of MPI tasks that collect the data from associated ranks and then write them to the disk in a serial manner, that is, data is written to the file immediately after is received from one of the associated ranks. This approach saves buffer memory but blocks the progress of the other associated ranks. The data layout depends upon ranks used in the computation, which in turn made the computation reconfiguration rather inflexible.
In other applications it was found that when IO is performed in this manner and especially for ASCII text files, the method loses scalability as the number of MPI ranks reaches the $1000 - 10000$ range. In order to ensure scalable IO when using more than 1000 MPI ranks, the original IO operations were modified as follow:

- Data from monitoring points will now be collected by a designated rank and then written as one text file. A post-processing utility program was written to split this file into separate files for the format used by the visualisation application. For the sake of portability (i.e. post-processing can be done on a local machine) the ASCII text format was preserved as the write time is not a large for the current runs, but it can be easily switched to a binary format if the writing time needs to be reduced.
- The IO operations for the restart file will now be done with MPI-IO in a single file. The main advantage of this is that the data is stored using the global grid order, thus allowing runs with a different grid decomposition to use the same restart file.

Also related to IO operations we mention that the logging mechanism was changed in order to avoid a multiple file access pattern. In the new version log messages are written to a single file by all MPI tasks using the shared file pointer provided by MPI-IO. For debugging purposes a subroutine that will dump the contents of the global arrays has also been provided along with a post-processing program for inspecting or comparing sections of dumped arrays.

The IO benchmark was carried out for three grid sizes, see Table 1, columns $4-8$. One can see that the write time of the monitoring data by using a single file is significantly improved, but it came as a bit of surprise that IO for the restart file is fastest when using Fortran IO with one writer per node, even for runs that used $18,432$ MPI ranks and approximately 217GB of checkpoint data. This result suggests that the best strategy for the restart operation is to use MPI-IO for the fine tuning of the run parameters (e.g. when searching for the best 2D MPI rank decomposition) and to switch to Fortran IO for the production runs, if this is faster.

5 Source code improvements

The modernisation of the legacy Fortan 77 source code to Fortran 95 was carried out following the standard steps described in literature [4]. The work carried out in this area included: replacing all common blocks with module variables, changing the `include` statement with the `use` statement, eliminating implicit typing and implicit interfaces and replacing the static arrays with allocatable versions.
Fortran 77 subroutines were wrapped in Fortran 95 modules, with one per file, according to their functionality. A brief description of the new modules is as follows:

- constant_and_kinds.f90: contains basic mathematical and code constants, data kind definitions for Fortran and MPI,
- global_data.f90: contains global data used across the source code, the needed entities are selected via the construct `use, global_data, only : ...`,
- commonvdrop.f90: stores data for the initial perturbation,
- basic_functions.f90 : mathematical functions used to compute physical properties across the other modules,
- plume.f90: contains the driver subroutine of the simulation, i.e. it controls the main time loop and associated procedures,
- pade_data.f90 : contains parameters used in the computation of the space derivatives,
- deriv3d_v601.f90 : contains subroutines that compute the derivatives in the x,y,z directions for the flow variables,
- main.f90: initialises MPI, reads the input file, calls the initialisation subroutines from modules that need data allocation/initialisation, calls the driver subroutine and finalisation operations,

- rhs3d.f90 : contains the subroutines that compute the flow variables and their derivatives at each grid point, boundary conditions and thermodynamical functions necessary for time integration,
- record3d.f90 : contains the subroutines that collect observation data during computation,
- restart.f90: contains the former common blocks used in restart subroutines and the interfaces for the implementation of MPI and Fortran IO restart algorithms,
- debug_tools.f90: contain subroutines that can be used to log warning or error messages to an unique file using a shared file pointer provided in MPI-IO.

During benchmarking and profiling it was noticed that a couple of subroutines of the core solver had a very poor cache memory utilisation. This problem was corrected by reordering some of the nested loops and by evicting `IF` blocks from some of the other loops. As a result the code performance in the RHS sector increased by approximately 20%, see Table 1.
Some other source code upgrades and reorganisations are listed below:

- Quasi-identical versions of some derivative subroutines were reduced to one version using an optional argument feature for module subroutines.
- The directory containing the source code was split into three directories; `src` for the source code, `utils` for post-processing programs and `decomp_2d` for the 2DECOMP library source.
- The Makefile was updated to allow compilation with different compilers, optimisation or in debug mode, and a full list of dependencies was created.
- User documentation for the changes has been provided in a README file.

The new code was compiled with PGI, GNU and Cray compilers using debug and optimised flags, all executables have been tested on several MPI processor topologies and grids with different aspect ratios.

6 Conclusion and future work

The upgraded version of DSTAR can use approximately 50 times more MPI tasks and grid points than the pre-dCSE version with good efficiency (in fact we have not touched the limit of scalability as of yet, due to practicalities of running very large jobs on HECToR). The

serial performance was improved significantly (approximately 20%), the parallel efficiency of the 2D decomposition has been improved by approximately 50%.
Both aspects (large scale parallelism and serial speed) are very important for the exploration of new physical regimes with DSTAR because the integration time step is controled by the micorscopical (chemical) time scale while the flow characteristics are determined by the macroscopic geometry and a much larger time scale.
The monitor data and debug or log messages are written to single files, a MPI-IO version for read/write of restart file has been provided which allows for flexibility, while the faster Fortran binary version was preserved.
The source code was reoganized into Fotran 90 modules, this makes the overall code structure clearer. In turn this helps to solve programing errors faster and offers the possibility to write simpler code from the top level subroutines. As an example of enhanced usability, we mention the case for the introduction of dynamical memory allocation for the data arrays which permits one to use the same executables for different grid sizes.
While working on the DSTAR code, it became apparent that even more could be done to improve DSTAR performance and usability.
As it was mentioned before, communication within the 2D domain decomposition consumes approximately 50% of the run time. A preliminary study shows that this can be decreased significantly by the introduction of a computation-communication overlap algorithm with the help of mixed mode programing. This concept could also be extended to the IO operations by creating a subset of IO nodes that execute IO operations concurrently with computation reserved for the other nodes.
From a usability perspective the following points will be considered for a future project: i) a structured input file that should provide information in three logical blocks: a) physical parameters, b) computational parameters and c) data collection parameters, ii) better user documentation for the input parameters and data post-processing, iii) an extended testing module, accessible as a Makefile task for validating new code developments and iv) an improved data collection with the help of the IO function from the 2DECOMP library.

Acknowledgements

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] J. Xia and K. H. Luo, *Conditional statistics of inert droplet effects on turbulent combustion in reacting mixing layers*, Combustion Theory and Modelling, 13:5, 901–920 (2009), and the references therein.

[2] `http://www.2decomp.org/`, see also N. Li and S. Laizet, *2DECOMP&FFT – A highly scalable 2D decomposition library and FFT interface*, Cray User Group 2010 conference, Edinburgh, UK; `http://www.hector.ac.uk/cse/distributedcse/reports/incompact3d/incompact3d/index.html`.

[3] L. Anton, N. Li and K. .H. Luo, *A study of scalability performance for hybrid mode computation and asynchronous MPI transpose operation in DSTAR* Cray User Group 2011 conference, Fairbanks, USA.

[4] D. Rouson, J. Xia and X. Xu, *Scientific Software Design: The Object Oriented Way*, Cambridge University Press, 2011.