Parallel I/O and Dynamic LES for Single Phase Flows

Luke Corcoran and Conor McCabe

June/July 2017

Abstract

S-TPLS is a fully parallelised code but makes use of serial I/O(Fannon et al., 2016). We implemented parallel I/O into this code, and tested both methods in order to compare scalability and to compute parallel efficiency. We found that the parallel I/O scaled more efficiently, with a consistently higher value for parallel efficiency E_p. We implemented a dynamic LES model into the code, an improvement over the previous Smagorinski model. We compared two dynamic models, and found that a localised model (Pomielli, 1995), was the most stable. However both dynamic models are found to be unstable for Reynolds numbers of interest ($\gtrsim$ 200) and thus a more stable localisation method is required. Finally, we implemented a new parameter input method in order to make the code more user-friendly. We combined these changes into a new version of the code, called AS-TPLS, written in FORTRAN-90.

1 Introduction and Theory

Two Phase Level Set (TPLS) is a fully parallelised three-dimensional Navier Stokes flow solver, which makes calls to a number of external libraries in order to boost computational speed. TPLS is written in Fortran90 and is parallelised using MPI and OpenMP message passing interfaces in order to facilitate scaling to thousands of CPU cores, and has been rigorously validated with respect to Orr-Sommerfeld, Orr-Sommerfeld-Squire, and Stuart-Landau semi-analytical theories (O'Naraigh et al., 2014).

Simplified Two Phase Level Set (S-TPLS) is a simplified version of the existing code created for pedagogical purposes in order to act as a replacement to the many "black box" approaches used in teaching introductory CFD (Fannon et al., 2016). Modelling only one flow, the second phase (i.e. the second fluid) has been removed to aid simplicity. S-TPLS retains many of the features of the original code, including parallelisation and good scalability, however it uses serial I/O as opposed to the parallel I/O framework of TPLS.

In creating Advanced Simplified Two Phase Level Set (AS-TPLS), we had three main goals: modify the existing Smagorinsky model to allow for dynamic calculation of the Smagorinsky coefficient, implement parallel I/O in order to alleviate bottlenecking issues and boost the scalability of the code, and to implement a parameter input method to make the code more user-friendly.

1.1 Problem statement

We consider an incompressible, Newtonian fluid confined the a channel geometry $\Omega = [0, L_x] \times [0, L_y] \times [0, L_z = 1]$(Figure 1), subject to a constant pressure drop $\frac{dp}{dx}$, so it is forced in the $x-$direction. Periodic boundary conditions are implemented in the $x-$ and $y-$directions, and a no-slip condition is enforced in the $z-$direction.

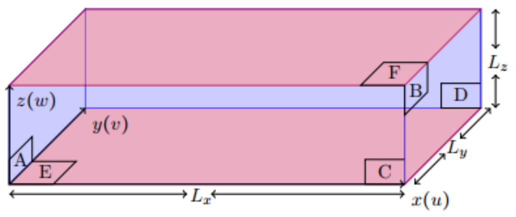

Figure 1: Computational Domain Ω

The governing equations which describe such a fluid are the Navier-Stokes equations, here written in non-dimensional form as follows:

$$\partial_t \vec{u} + (\vec{u} \cdot \vec{\nabla})\vec{u} = -\vec{\nabla} p + \frac{1}{Re}\nabla^2 \vec{u} \tag{1}$$

$$\vec{\nabla} \cdot \vec{u} = 0 \tag{2}$$

where $\vec{u} = (u, v, w) \equiv (u_1, u_2, u_3)$ is the velocity field of the fluid and p is the pressure field. $Re = \frac{\rho U L}{\mu}$ is the Reynolds number which characterises the flow, with ρ the density of the fluid, U, L the velocity and length scales respectively, and μ the dynamic viscosity. In order to solve these equations numerically, the fields are discretised on a marker-and-cell (MAC) grid, whereby velocity values are stored at the edges of the cells, and the scalar valued pressure and viscosity fields and stored at cell centres(Figure 2). This leads to a natural way of interpreting derivatives between cell faces, consistent with the divergence theorem. This computational grid induces a natural filter on the fields. Given a field variable $f(\vec{x}, t)$, its filtered counterpart $\bar{f}(\vec{x}, t)$ can be written:

$$\bar{f}(\vec{x}, t) = G \star f \equiv \int G(\vec{y}, \vec{x}, \Delta) f(\vec{x} - \vec{y}, t) d\vec{y} \tag{3}$$

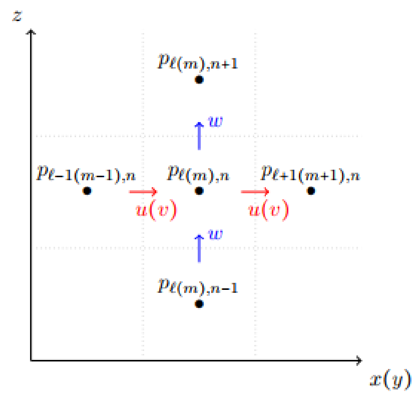

Figure 2: MAC Grid

where G is a convolution kernel, with characteristic length scale Δ, which vanishes at infinity in momentum space. This ensures that Fourier modes above a certain wavenumber are suppressed in the filtered variables. Application of the filtering operation (3) to (1) and (2) leads to the filtered equations of motion, written now in component form:

$$\partial_t \bar{u}_i + \bar{u}_j \partial_j \bar{u}_i = -\partial_i \bar{p} + \frac{1}{Re}\partial^2 \bar{u}_i - \partial_j \tau_{ij} \tag{4}$$

$$\partial_i \bar{u}_i = 0 \tag{5}$$

where $\tau_{ij} = \overline{u_i u_j} - \bar{u}_i \bar{u}_j$ is the residual stress tensor and represents the contribution of the small scale motions of the fluid. (4) and (5) cannot yet be explicitly solved, due to the presence of the $\overline{u_i u_j}$ term in (4). The procedure of modelling these small scale motions is known as 'large eddy simulation'(LES) and there are various ways to do this.

1.2 The Dynamic Smagorinski Model

In order to make any progress in solving (4) and (5), τ_{ij} needs to be modelled on physical grounds. The simplest model for this residual stress tensor is declaring its traceless part to be proportional to the local filtered strain rate $\bar{s}_{ij} = \frac{1}{2}(\partial_j \bar{u}_i + \partial_i \bar{u}_j)$, with a constant of proportionality being the viscosity of sub-grid scale eddies ν_t:

$$\tau^d_{ij} \equiv \tau_{ij} - \frac{1}{3}\delta_{ij}\tau_{kk} = -2\nu_t \bar{s}_{ij} \tag{6}$$

This eddy viscosity is modelled:

$$\nu_t = C_s \bar{\Delta}^2 |\bar{\mathbf{s}}| \tag{7}$$

where $C_s > 0$ is the so-called Smagorinski co-efficient, $\bar{\Delta}$ is the filter scale, and $|\bar{\mathbf{s}}| = \sqrt{2\bar{s}_{ij}\bar{s}_{ij}}$. With these assumptions, (4) becomes:

$$\partial_t \bar{u}_i + \bar{u}_j \partial_j \bar{u}_i = -\partial_i \bar{p} + 2\partial_j (\frac{1}{Re} + \nu_t)\bar{s}_{ij} \tag{8}$$

This model, dubbed the Smagorinski model, closes equations (4) and (5) and has proven to be very successful in modeling fluids. However, there is little justification for taking C_s to be a positive constant in (7). This model fails to predict backscatter, the transfer of energy from smaller to larger eddies, and near-wall terms need to be included to ensure correct behaviour of ν_t near boundaries. The dynamic Smagorinski model (DSM) aims to fix this. It is based on introducing a test filter on top of the grid filter, with scale $\tilde{\Delta} = \alpha\bar{\Delta} > \bar{\Delta}$, where $\alpha = 2$ is found to be the best value of α. This filtering leads to a test residual stress tensor $T_{ij} = \widetilde{\overline{u_i u_j}} - \tilde{\bar{u}}_i\tilde{\bar{u}}_j$, analogous to τ_{ij} in (4). The DSM is built from the premise that C_s, while spatially and temporally varying, should be independent of the filter width. Thus we model T_{ij}:

$$T_{ij}^d \equiv T_{ij} - \frac{1}{3}\delta_{ij}T_{kk} = -2\tilde{\nu}_t\tilde{\bar{s}}_{ij} = -2C_s\tilde{\Delta}^2|\tilde{\bar{\mathbf{s}}}|\tilde{\bar{\mathbf{s}}}_{ij} \tag{9}$$

Introducing $L_{ij} = \widetilde{\bar{u}_i\bar{u}_j} - \tilde{\bar{u}}_i\tilde{\bar{u}}_j$ it is apparent that:

$$L_{ij}^d \equiv L_{ij} - \frac{1}{3}\delta_{ij}L_{kk} = T_{ij}^d - \tilde{\tau}_{ij}^d \tag{10}$$

if we allow C_s to pass through the test filtering operation. If we define $M_{ij} = -2\bar{\Delta}^2\widetilde{|\bar{\mathbf{s}}|\bar{\mathbf{s}}_{ij}} - 2\tilde{\Delta}^2|\tilde{\bar{\mathbf{s}}}|\tilde{\bar{\mathbf{s}}}_{ij}$ then the following equation holds for C_s:

$$L_{ij}^d = C_s M_{ij} \tag{11}$$

To find the best value of C_s in the dynamic model, we appeal to two different methods. The first (DSM1) is by minimising the mean-square error of the approximation in (11), which leads to the specification:

$$C_s = \frac{M_{ij}L_{ij}}{M_{kl}M_{kl}} \tag{12}$$

As the values for M_{ij} and L_{ij} fluctuate and can be very large, C_s is averaged along homogeneous flow directions for stability. The second method (DSM2) was proposed by Pomielli(1995), and takes note of a mathematical inconsistency which arises in the derivation of (12), namely the fact the we implicitly assumed a spatial independence of C_s in deriving (10), as it is passed through the test filtering operation. Without this assumption, (10) becomes:

$$L_{ij}^d = -2\tilde{\Delta}^2 C_s|\tilde{\bar{\mathbf{s}}}|\tilde{\bar{\mathbf{s}}}_{ij} + 2\bar{\Delta}^2\widetilde{C_s|\bar{\mathbf{s}}|\bar{\mathbf{s}}_{ij}} \equiv -2C_s\alpha_{ij} + 2\widetilde{C_s\beta_{ij}} \tag{13}$$

Denoting the C_s under the tilde by C^*, our best guess for C_s at the current time step, (12) is solved for C_s:

$$C_s\alpha_{ij} = -\frac{1}{2}(L_{ij}^d - \widetilde{C^*\beta_{ij}}) \tag{14}$$

Again, minimising the mean-square error of (13) leads to an expression for C_s:

$$C_s = -\frac{1}{2}\frac{(L_{ij}^d - \widetilde{C^*\beta_{ij}})\alpha_{ij}}{\alpha_{kl}\alpha_{kl}} \tag{15}$$

Upon calculation of C_s for DSM1 (12) or DSM2 (15), (6) can be used to find τ_{ij} and (4) and (5) may be solved. DSM2 is the method used in AS-TPLS.

2 Parallel I/O Implementation

Dr. O'Naraigh provided a template parallel I/O framework which had been used in the original TPLS code (O'Naraigh et al., 2014). This fit quite well into the S-TPLS code and needed only minor adjustment in order to be run successfully. The parallel I/O makes use of NetCDF, a data storage system which supports array oriented scientific data. Storing each output variable as a 3 dimensional array cuts down on post-processing of data and aids usability. We inserted a logical flag into the code to allow for easy choice between serial and parallel I/O.

We ran into problems when trying to compile the code in an environment without NetCDF support. This was remedied by creating a new module in which to store the NetCDF subroutines and by commenting out the two remaining references to NetCDF in the main code, i.e. 'use netcdf' and 'use netcdf_stuff'. We then created a file with empty subroutines of the same name which must be linked to when compiling. This is a somewhat inelegant solution and might be improved upon in further work by making use of Fortran preprocessor macros.

Number of CPU Cores	Parallel I/O	Serial I/O
8	503.90	594.93
32	187.02	289.15
50	147.60	247.52
128	98.92	198.16
200	91.78	187.50

Table 1: Time taken in seconds to run $1,000$ iterations on varying CPU core counts.

The AS-TPLS code was run with a grid $N_x = 161$, $N_y = 81$, and $N_z = 81$ for a total grid count of $N = 1.056 \times 10^6$ for $1,000$ time iterations, on various numbers of processors.

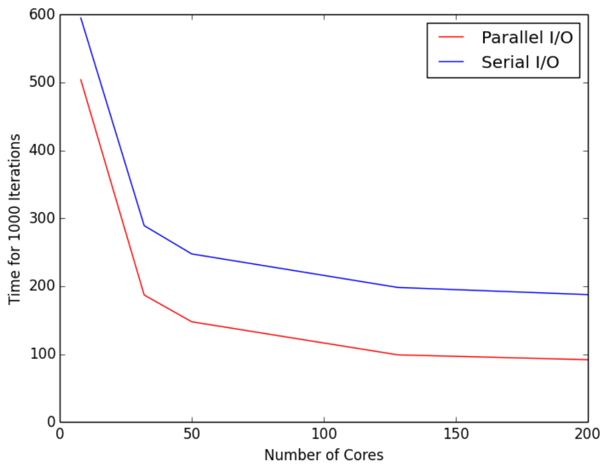

Figure 3: Comparison of Parallel/Serial I/O Speeds

In order to increase the usability of the code, we implemented a new parameter input method. Prior to this, users needed to hard code parameters which may have hindered the code's use as a pedagogical tool. All parameters are now entered into a text file and read from there into the main code.

We calculated and plotted the parallel efficiency, a measure of scaling efficiency, according to the following formula:

$$E_p = \frac{e_p}{e_8} = \frac{T_1}{pT_p}\frac{8T_8}{T_1} = \frac{8T_8}{pT_p}$$

Where T_1 is the run time on 1 process and T_p is the run time on p processes. We used 8 cores as the base case because it was not feasible to run the program on a single core.

Optimal parallel efficiency is 1, thus it is clear that parallel I/O scales more efficiently than serial I/O at all core counts.

Number of CPU Cores	Parallel I/O	Serial I/O
32	0.67	0.514
50	0.546	0.385
128	0.318	0.188
200	0.22	0.127

Table 2: Parallel efficiency of parallel and serial I/O.

3 Dynamic LES Implementation

In order to apply the test filtering operation to the velocity fields, a subroutine was written to calculate the ‘cell-centred’ velocities u_c, v_c and w_c. The details of the MAC grid are not covered here, but due to the indexing used the cell centred velocities were calculated by the following simple averages:

$$u_c(i,j,k) = \frac{1}{2}(u(i,j,k-1) + u(i-1,j,k-1)) \tag{16}$$

$$v_c(i,j,k) = \frac{1}{2}(v(i,j,k-1) + v(i,j-1,k-1)) \tag{17}$$

$$w_c(i,j,k) = \frac{1}{2}(w(i,j,k) + w(i,j,k-1)) \tag{18}$$

A subroutine was written to calculate the fields necessary in the implementation of DSM, namely the product $\bar{u}_i\bar{u}_j$ (calculated with cell centred values), $\bar{s}_{ij}$, $|\bar{s}|$, $|\bar{s}|\bar{s}_{ij}$, and $C^*\beta_{ij}$ (for DSM2). Each of these arrays is populated locally on each MPI process. Halo swaps are performed between neighbouring processors for taking derivatives/filtering. The test filter, with double the width of the grid filter, is realised by nearest-neighbour averaging of the field variables. Explicitly: given $\bar{f}(i,j,k)$, its doubly filtered counterpart $\tilde{\bar{f}}(i,j,k)$ is:

$$\tilde{\bar{f}}(i,j,k) = \frac{1}{6}(\bar{f}(i+1,j,k)+\bar{f}(i-1,j,k)+\bar{f}(i,j+1,k)+\bar{f}(i,j-1,k)+\bar{f}(i,j,k+1)+\bar{f}(i,j,k-1)) \tag{19}$$

Using (19) the doubly filtered fields used in the calculation of C_s can be calculated using (12) or (15). In DSM1 C_s is averaged along homogeneous directions of the flow in each MPI process, leading to a piecewise constant viscosity in the channel. C_s in DSM2 is not averaged, the value of C^* is taken as C_s at the previous time step. There are various more complicated, iterative ways to pick C^*, as described in [2]. With τ_{ij} modelled the Navier-Stokes equations are then solved as described in [1].

The DSM1 and DSM2 algorithms were found to be unstable for Reynolds numbers of interest (Figure 1). In [1] the Smagorinski model was used to simulate a single phase channel flow for $Re = 360$, whereas DSM1 and DSM2 blew up very quickly at this value of Re.

This ‘blowing up’ manifested itself as an instability in the pressure solver. Due to higher stability at lower Reynolds numbers and more mathematical consistency DSM2 was used in the final version of AS-TPLS, however stability at higher Reynolds numbers is greatly desired.

Although unstable, DSM2 does provide reasonably plots for the u-velocity and viscosity fields after 300,000 time iterations at $dt = 10^{-4}$. Figure 5 shows a slice of u-velocity. This plot looks reasonable due to the pressure drop in the x-direction, and thus the x-component of the velocity field should have a high value away from the walls. This nice plot may however be an artifact of the suggestive initial conditions used for the velocity field. Figure 6 shows a slice of the viscosity field, which shows a vanishing viscosity at the boundaries and small patches of high viscosity which we may interpret as eddy currents.

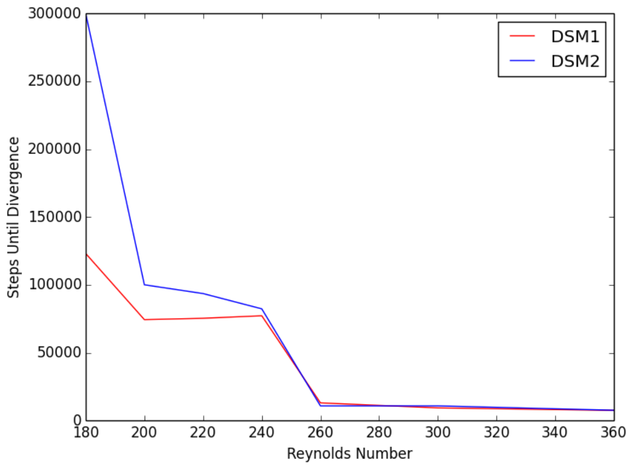

Figure 4: Comparison of DSM1 and DSM2 stability

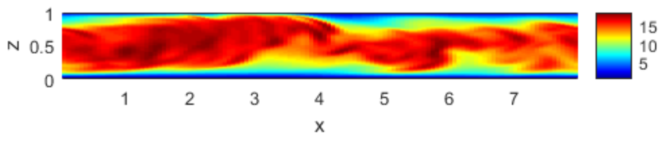

Figure 5: DSM2: Slice of u-velocity in xz-plane at y=midpoint, 300,000 iterations

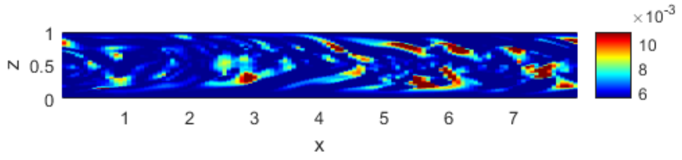

Figure 6: DSM2: Slice of viscosity in xz-plane at y=midpoint, 300,000 iterations

4 Conclusions and Future Work

The new version of S-TPLS, AS-TPLS, was successfully produced. This version includes parallel I/O implementation, a dynamic LES concept, and a user-friendly input method. The parallel I/O was found to speed up the code and also scales more efficiently than serial I/O. Two dynamic models were tested, and both were found to be unstable for Reynolds numbers of interest. Thus it is of great interest to find a way to stabilise these models. One way of realising this might be to implement a 'mixed' dynamic model, such as the one proposed by Bardina and implemented by Zang et al. in 1993. This model is reported to be the most successful LES model by Vreman et al. in 1997 in their investigation of turbulent mixing layers. It would be interesting to stabilise any of these models since none of these avail of the parallel computing techniques employed here.

5 Appendix - Some notes on running the code

In order to change the input parameters, the file "input.txt" should be edited. There are further instructions in the file regarding what format the data should take.

The submit script we used to run the code on Fionn[1] was:

```
module load dev intel/2015/-u3

module load libs netcdf/intel_mpi/4.3.0

mpiifort -c netcdf_stuff.f90
-I/ichec/packages/netcdf/intel_mpi/4.3.0/include
-L/ichec/packages/netcdf/intel_mpi/4.3.0/lib
-lnetcdff -lnetcdf

mpiifort -O3 final_main_spsrj_LES.f90
ds_momentum_stuff_allflux_LES.f90 mpi_stuff.f90
pressure_stuff.f90 sor_iteration_allflux_spsrj.f90
sphase_LES_initialisation.f90 -o stpls.x ./netcdf_stuff.o
-I/ichec/packages/netcdf/intel_mpi/4.3.0/include
-L/ichec/packages/netcdf/intel_mpi/4.3.0/lib
-lnetcdff -lnetcdf
```

Future users should note that the "netcdf_stuff.f90" file which contains the NetCDF subroutines is specified as a module which means it needs to be compiled separately to the main file and then linked to in the main compile line.

If you want to compile in an environment without NetCDF support, you will need to include an additional link to the file "no_netcdf_support.f90" which contains the empty subroutines.

An example compile script used to successfully compile the code on Orr[2] is:

```
/share/apps/mvapich2gnu/bin/mpif90 -O3
final_main_spsrj_LES.f90 mpi_stuff.f90
sphase_LES_initialisation.f90 pressure_stuff.f90
ds_momentum_stuff_allflux_LES.f90
sor_iteration_allflux_spsrj.f90 no_netcdf_support.f90
-o stpls.x
```

[1]Fionn is the Irish supercomputer cluster. More information at https://www.ichec.ie.
[2]Orr is UCD's local computer cluster.

6 References

[1] - Fannon J, Loiseau J-C, Valluri P, Bethune I and Ó Náraigh L 2016 High-performance computational fluid dynamics: a custom-code approach *Eur. J. Phys.* **37**
[2] - Piomelli U Large-eddy simulation of rotating channel flows using a localized dynamic model 1995 *Physics of Fluids* **7**, 839
[3] - Ó Náraigh L, Valluri P, Scott D M, Bethune I, Spelt P D M 2014 Linear instability, nonlinear instability, and ligament dynnamics in three-dimensional laminer two-layer liquid flows *J. Fluid Mech.* **750** 464-506
[4] - TPLS: High Resolution Direct Numerical Simulation of Two-Phase Flows. http://sourceforge.net/projects/tpls/
[5] - Zang, Y., Street, R.L. and Koseff, J.R. (1993) A Dynamic Mixed Subgrid-Scale Model and Its Application to Turbulent Recirculating Flows. Physics of Fluids A: Fluid Dynamics, 5, 3186-3196.
[6] - Large-eddy simulation of the turbulent mixing layer, Bert Vreman, Bernard Geurts and Hans Kuerten.
[7] - AS-TPLS Source Code. https://gitlab.com/zagaluke/AS-TPLS