*September 2015*

# Developing a scalable and flexible high-resolution DNS code for two-phase flows

Iain BETHUNE [a,1], Antonia B K COLLIS [a], Lennon Ó NÁRAIGH [b], David SCOTT [b] and Prashant VALLURI [c]

[a] *EPCC, The University of Edinburgh, UK*
[b] *School of Mathematical Sciences and CASL, University College Dublin, Ireland*
[c] *Institute for Materials and Processes, School of Engineering, The University of Edinburgh, UK*

**Abstract.** We introduce TPLS (Two-Phase Level Set), an MPI-parallel Direct Numerical Simulation code for two-phase flows in channel geometries. Recent developments to the code are discussed which improve the performance of the solvers and I/O by using the PETSc and NetCDF libraries respectively. Usability and functionality improvements enabled by code refactoring and merging of a separate OpenMP-parallelized version are also outlined. The overall scaling behaviour of the code is measured, and good strong scaling up to 1152 cores is observed for a 5.6 million element grid. A comparison is made between the legacy serial text-formatted I/O and new NetCDF implementations, showing speedups of up to 17x. Finally, we explore the effects of output file striping on the Lustre parallel file system on ARCHER, a Cray XC30 supercomputer, finding performance gains of up to 12% over the default striping settings.

**Keywords.** Computational Fluid Dynamics, Direct Numerical Simulation, PETSc, NetCDF, Parallel I/O

## Introduction

TPLS (Two-Phase Level Set) is an open-source program for simulation of two-phase flows in 3D channel geometries using high resolution Direct Numerical Simulation. Due to the high computational cost of these calculations, parallelization is essential, and scaling has now been demonstrated to several thousand CPU cores. To achieve this, the code has been developed by a collaboration of experts in HPC applications development, applied mathematics and algorithms, and the physics of multiphase flows.

TPLS is unique in several aspects. Unlike other programs such as OpenFOAM, the TPLS solver has been purpose-built for supercomputing architectures like ARCHER and keeping large scale simulations of two-phase interfacial flows (coupled with complex transport phenomena) in mind. Most open source solvers like OpenFOAM, Gerris, Fluidity and commercial solvers like ANSYS-Fluent/CFX offer only one interface-capturing

[1] Corresponding Author: *ibethune@epcc.ed.ac.uk*

method (the volume-of-fluid method) thereby limiting the applicability of these solvers to either free-surface, stratified, or wavy-stratified flows. TPLS offers the users a choice of two types of interface-capturing methods: the diffuse-interface method and the levelset method. This enables the solver to accurately simulate not only the aforementioned flows but also a wide variety of physics (currently unavailable in all of the above solvers) including stratified-slug flow transitions, interfacial turbulence [1], counter-current flows [2], contact-line motion, phase change and heat transfer [3].

## 1. TPLS Overview

TPLS solves the incompressible Navier-Stokes equations. The interface is captured using either a choice of levelset [4] or diffuse-interface [5] methodologies (the latter is referred to hereafter as DIM). Detailed equations for the implementation of the levelset framework are presented below; the reader is referred to Reference [5] for the details concerning the implementation of the DIM methodology.

In the levelset formulation, a continuous surface tension model [4] is employed as a model for the two-phase Navier–Stokes equations with the sharp interfacial matching conditions. The model equations for the velocity $\boldsymbol{u}$ and the pressure $p$ read

$$\rho(\phi)\left(\frac{\partial \boldsymbol{u}}{\partial t}+\boldsymbol{u}\cdot\nabla\boldsymbol{u}\right)=-\nabla p+\frac{1}{\mathrm{Re}}\nabla\cdot\left[\mu\left(\nabla\boldsymbol{u}+\nabla\boldsymbol{u}^{T}\right)\right]+\frac{1}{\mathrm{We}}\delta_{\varepsilon}(\phi)\hat{\boldsymbol{n}}\nabla\cdot\hat{\boldsymbol{n}}+\mathscr{G}\rho(\phi)\widehat{\boldsymbol{z}}, \tag{1a}$$

$$\nabla\cdot\boldsymbol{u}=0, \tag{1b}$$

$$\hat{\boldsymbol{n}}=\frac{\nabla\phi}{|\nabla\phi|},\qquad \frac{\partial\phi}{\partial t}+\boldsymbol{u}\cdot\nabla\phi=0, \tag{1c}$$

where $\widehat{\boldsymbol{z}}$ is the unit vector prescribing the downward-direction of the acceleration due to gravity. Here also, $\phi(\boldsymbol{x},t)$ is the levelset function indicating in which phase the point $\boldsymbol{x}$ lies ($\phi<0$ in the bottom layer, $\phi>0$ in the top layer). The (possibly multivalued) interface $\eta(\boldsymbol{x},t)$ is therefore the zero level set, $\phi(\boldsymbol{x},t)=0 \implies \boldsymbol{x}=(x,y,\eta(x,y,t))$. Moreover, the levelset function determines the unit vector normal to the interface ($\hat{\boldsymbol{n}}$) as well as the density and viscosity, via the relations $\rho=r(1-H_{\varepsilon}(\phi))+H_{\varepsilon}(\phi)$ and $\mu=\mathrm{Re}^{-1}\left[m(1-H_{\varepsilon}(\phi))+H_{\varepsilon}(\phi)\right]$ respectively. The function $H_{\varepsilon}(\phi)$ is a regularized Heaviside function, which is smoothed across a width $\varepsilon=1.5\Delta x$ ($\Delta x$ is the grid spacing, see below). Finally, $\delta_{\varepsilon}(s)=dH_{\varepsilon}(s)/ds$ is a regularized delta function supported on an interval $[-\varepsilon,\varepsilon]$. Note that the model is presented in non-dimensional form, such that Re is the Reynolds number, $1/\mathrm{We}$ is a dimensionless measure of surface tension (We is the Weber number), and $\mathscr{G}$ is a dimensionless version of the gravitational constant.

Equations (1) are solved numerically on a regular 3D grid of grid spacing $\Delta x$ in each Cartesian direction. A finite-volume discretization is employed based on an idealised channel geometry with a range of different inlet conditions that can be prescribed by the user (Figure 1). The code evolves the physical variables (pressure, fluid velocities, and interface configuration) through discrete time steps. The pressure is treated using a projection method and the time marching is carried out using a combination of third-order

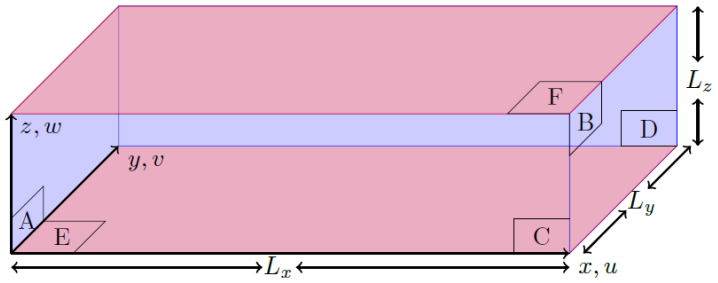


**Figure 1.** (Picture thanks to James Fannon). Computational domain $[0, L_x] \times [0, L_y] \times [0, L_z = 1]$. Periodic boundary conditions (PBC) or Neuman conditions are used in the streamwise (faces AB). PBC are used in the spanwise (faces CD) directions. Finally, no-slip boundary conditions are used on faces $E$ and $F$.

Adams-Bashforth and second-order Crank-Nicholson methodologies. At each time step, the key computational tasks performed amount to the solution of large systems of sparse linear equations with tens of millions of unknowns, for the key physical variables. In addition regular I/O is required to save the system state for later analysis and visualization, or restart in the case of hardware failure.

The code is implemented in Fortran 90, initially with MPI parallelization using a 2D domain decomposition (in $x$ and $y$ dimensions) and bespoke Jacobi / SOR iterative solvers. Over the last two years, we have improved the TPLS code in several respects to give better performance, scalability and usability, moving from an in-house code specialised for use by the original developers, to a open-source, flexible program which can be easily be used by others, including academic and industrial users.

### *1.1. Linear solvers from PETSc*

TPLS was first released in June 2013 on Sourceforge [6] under the open-source BSD licence. With support from the HECToR Distributed Computational Science and Engineering (dCSE) scheme, we re-implemented the most expensive part of the calculation – the pressure solver [7] – in PETSc [8].

We make use of two main features of PETSc. A Distributed Array (DA) object is used to represent the distributed regular 3D grid described in Figure 1, holding a scalar value (the pressure) at each grid point. At present, for compatibility with the rest of the code which does not yet use PETSc, we retain the same 2D domain decomposition to avoid an expensive data redistribution before and after the pressure solve, requiring only a local copy-in/copy-out operation on each process. In general PETSc permits all three dimensions to be decomposed and eventually once all solvers in TPLS have been migrated to use PETSc this could be used to give additional scalability since for the same number of processors, a 3D decomposition results in more compact domains with smaller surface area resulting in reduced communication. The DA also provides the periodic boundary conditions in the spanwise direction, without the need for adding explicit halo swapping code.

To compute the pressure at a given time step we construct a PETSc distributed sparse matrix $\boldsymbol{A}$, representing the discretized equation in a canonical $\boldsymbol{A}x = b$ matrix-vector equa-

tion form. PETSc provides a KSP (Krylov SubsPace) object, which implements various iterative algorithms for solving linear equations of this form, abstracting away the details of the solution algorithm including the implementation of communication and periodic boundary conditions, and allows access to a wide variety of different Krylov solvers and preconditioners. Initial tests using the GMRES method with a Block Jacobi preconditioner, showed a speedup of 54% in the pressure solve on 1024 cores, using a grid with 11 million points, and 80% on 2048 cores, compared with the legacy Jacobi/SOR iteration (see [7] for timing data). This is due to improved strong scaling, rising from 60% efficiency from 1024 to 2048 cores with the Jacobi/SOR method, to linear scaling (100% efficiency) with the PETSc implementation. At this stage, the major remaining bottleneck was output, which on 1024 cores could take around 50% of the total runtime.

### *1.2. NetCDF I/O*

In TPLS version 2.0, we have re-implemented the original serial output routines using the NetCDF library [9]. All of the state variables in TPLS are stored on distributed 3D grids. For ease of post-processing, this distributed data must be reconstructed in canonical Cartesian order and written to a single file, which initially contained the data in a text format. TPLS version 1.0 used a 'single file, single writer' or 'master IO' I/O approach, that is, a 'master' process coordinated the I/O, receiving multiple MPI messages from all other processes, rearranged the data to reconstruct the global data set, and then performed the file write. Despite being reasonably straightforward to implement, this method has several disadvantages; other processes will be idle while the master performs I/O (or will idle after arriving early at the next synchronization point), the I/O time is a constant so will inhibit strong scaling due to Amdahl's law, and the need to store the global data limits the size of problem which can be studied to that which fits in memory on a single process. With this master I/O structure, each MPI process, sends 105 messages to the master process, which adds significant overhead.

Three types of output files were generated by TPLS 1.0: `phi-channel`, `uvw-channel` and a backup file. The file `uvw-channel` contains four variables — `U, V, W, Pressure` — each labelled by grid coordinates, which are then separated during post-processing. The backup or checkpoint file contains all 14 state variables amalgamated into one single file. In the new I/O strategy, these files were split up so that one variable is stored per file. This was found to improve performance as although many small files will limit the optimal use of bandwidth, one large file results in excessive synchronization between processes.

NetCDF provides various routines for parallel I/O, allowing all processes to write simultaneously to a single file, and stores data in a binary file format with optional compression using the HDF5 library [10]. While HDF5 does not support writing to compressed files in general, when using HDF5 behind the NetCDF interface data is written with a defined 'chunk size' enabling the HDF5 compression algorithm to work correctly in parallel.

TPLS uses the following command from the NetCDF Fortran 90 interface to create the new I/O files:

```
nf90\_create(filename, cmode, ncid,
     comm=PETSC\_COMM\_WORLD, info=MPI\_INFO\_NULL)
```

The use of the comm and info parameters ensure that parallel I/O is enabled. The variable `cmode` is set to:

```
cmode = IOR(NF90\_NETCDF4, NF90\_MPIIO)
```

which results in the creation of a parallel HDF5/NetCDF-4 file, which will be accessing using MPI-IO.

By default parallel file access using NetCDF is 'independent', in that any process may access the file without waiting for others, also known as 'single file, multiple writers'. In this mode, files are created by each process individually writing the appropriate section of a file, according to the defined data distribution (see below). In this setup data is not transferred between processes, but synchronization is required. The alternative is a 'collective' file access pattern where each process must communicate with all other processes to receive and rearrange data but the data is written all at once. The 'independent' I/O model results in file locking when each process writes and therefore it is expected that this method does not scale. With the 'collective' method, i.e. the I/O system knows that all processes are writing, enabling optimization of the I/O operation specific to the I/O pattern and potentially increasing bandwidth. This pattern of I/O should result in the best scaling for file writes and is the choice implemented in TPLS.

For convenience, we have retained the original I/O routines as an option which can be selected through the use of input file parameters to enable the use of TPLS on machines where NetCDF is not installed and/or not desired. We also implemented full control of the I/O frequency for each file type independently and also the frequency of the restart checkpointing which was previously tied to the hard-coded frequency of regular file writing. Removing the need to provide checkpoint files at each data output step in itself can provide significant speedups for users.

As a result, we have obtained an order-of-magnitude reduction in I/O time, a compression factor of 6.7 compared to the TPLS 1.0 text files and removed the memory bottleneck of requiring rank 0 to gather the entire domain. We have also refactored the original routines which wrote periodic backups of the simulation state for restarting a crashed simulation to use the same file format as the output produced for analysis, reducing code complexity significantly.

We provide convenient post-processing tools to convert between the text and NetCDF file format, to aid data analysis. However, we note that NetCDF can be read natively by many visualization and analysis tools, including ParaView, which can easily produce iso-surfaces (e.g. Figure 2), volume renderings, and movies from TPLS output.

### *1.3. Diffuse Interface Method*

In addition to the Level Set method, we have added an MPI-parallel implementation of the Diffuse Interface Method (DIM), which is available as an option to users. The DIM is based on the Cahn-Hilliard equation [11] which describes phase separation. The DIM was originally developed by Pedro Sáenz [3] in a separate code which was used to study droplets. This code featured only OpenMP parallelism and so could not scale to the large core counts needed for physically relevant system sizes of the channel geometries supported by TPLS. Importing this code into TPLS involved establishing the relationships between the variables (and the way they are indexed) in the two codes and changing the boundary conditions to match those in Figure 1. One particular point to note is that

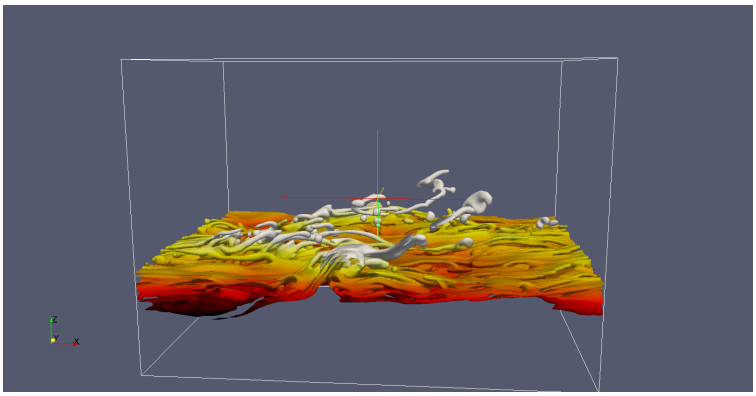

**Figure 2.** Sample output from TPLS using Paraview visualization software, showing the isosurface $\phi = 0$, coloured by the vertical elevation

the value of order parameter $\phi$ used to characterize the phase lies between 0 and 1 in the levelset implementation but between -1 and +1 in DIM. The Cahn-Hilliard equation is a fourth order equation. It was implemented as two coupled second order equations each of which has a stencil size of one. This helped the integration of the DIM code as TPLS already used halo swaps appropriate to this stencil size. Distributing the loops over the indices of the variables across multiple MPI processes, when before they had been confined to one process (because of the use of OpenMP) was straightfoward within this framework.

### *1.4. Usability Improvements*

Finally, with the support Mike Jackson of the Software Sustainability Institute, we have completely redesigned the user interface to TPLS. All simulation parameters can now be configured through input files or command-line arguments, obviating the need for users to modify and recompile the code for every application. A separate tool for creating initial flow conditions needed to start a new simulation is also provided, making use of NetCDF as the file format, again removing the need to modify the hard-coded initialization routines. The code has also been refactored to improve extensibility, and we used this to allow the DIM method to be provided as an option to the user. In future, this will allow for features such as different simulation geometries (e.g. counter-current flows) and new physics (e.g. phase change) which are currently implemented in separate programs to be included in the main TPLS version.

## 2. Benchmark results

To demonstrate the performance characteristics of TPLS, we have carried out strong scaling benchmark calculations using a grid size of 256x144x152, or 5.6 million elements. This is smaller than previous studies [7], but serves to expose the strong scaling behaviour at smaller numbers of cores than would be apparent with a larger grid. To be representative of production calculations we have executed 5,000 time steps, with paral-

lel output every 1,000 steps. The timings reported were measured on ARCHER, a Cray XC30 supercomputer with two 12-core 2.7 GHz Intel Ivy Bridge CPUs per node, and an Aries interconnect network connecting the total of 4920 compute nodes.

**Table 1.** Scaling of TPLS using Levelset method

| Nodes | Cores | Wallclock time (min) | Speedup |
|---|---|---|---|
| 1 | 24 | 675 | 1.00 |
| 6 | 144 | 82 | 8.23 |
| 12 | 288 | 38 | 17.8 |
| 48 | 1152 | 17 | 39.7 |

Table 1 shows superlinear scaling up to 288 cores, due to the local sections of the distributed 3D grids becoming small enough to fit fully within the processor cache hierarchy. At 1152 cores communication overhead becomes evident although the scaling efficiency is still over 80%. Because of the 2D decomposition, each process has a long, narrow domain of size 8x4x152 and so the communication to computation ratio is high since the quantity of halo data to be exchanged scales with the surface area of the domain rather than the interior volume. Complete conversion of the code to use PETSc, enabling use of a 3D domain decomposition will improve this as the domains will become more compact, reducing the impact of communication for a fixed number of processes.

For this grid size on 1152 cores, TPLS has around 5000 elements per core. This is similar to or slightly better than other open-source codes such as OpenFOAM, where the optimal number of cores for similar problem sizes was found to be range from 128 to 1024 depending on which physics and geometry were employed (see [12], table 6). By contrast, the commercial CFD code HyperWorks Virtual Wind Tunnel, achieves only 80% scaling efficiency on 576 cores, using a much larger mesh (22 million elements, 38000 per core) [13].

## 3. Parallel I/O

As discussed in section 1.2, the changes to the I/O routines improve performance, decrease the total file size by use of HDF5 compression and facilitate ease of use by providing the user more controls over types and frequency of output. This section discusses in more detail the performance improvements observed.

### *3.1. Performance of the NetCDF I/O Implementation*

Table 2 shows the performance improvement in the total execution (wallclock) time of the TPLS code from the old text formatted 'single file, single writer' mode to the 'single file, collective write' mode enabled by the use of parallel NetCDF. The times reported are means from 3 runs of a 100 time step simulation on a 512x144x152 grid, writing output and backup files every 10 time steps. These are more frequent than would be typical for a production calculation (as in Section 2), and the large grid size is designed to increase the cost of I/O. This comparison was performed after the code refactoring that whilst

**Table 2.** Performance of TPLS using NetCDF and text I/O routines

| **Number of nodes** | **1** | **3** | **24** | **128** | **256** |
|---|---|---|---|---|---|
| **Number of processors** | **24** | **72** | **576** | **3072** | **6144** |
| Text time (s) | 10211 | 7589 | 7347 | 7530 | 11879 |
| NetCDF time (s) | 3998 | 1838 | 428 | 689 | 1635 |
| Relative Speedup | 2.6 | 4.1 | 17.1 | 10.9 | 7.3 |

leaving the original I/O routines intact, improved control of the routines so that a like-for-like comparison could be made using the same frequency of file output for both text and NetCDF modes. These results show that for an 11 million grid point simulation the NetCDF implementation improves the runtime of TPLS by a factor of between 2.6 and 17.1 depending on the number of cores used. The best relative speedup between the two I/O methods is obtained on 24 nodes (576 cores) of ARCHER. Some of the speedup may be attributed to the reduction in the total amount of data to be written (a factor of 6.7), however this clearly does not explain the total improvement on core counts above 72 is greater than 6.7.

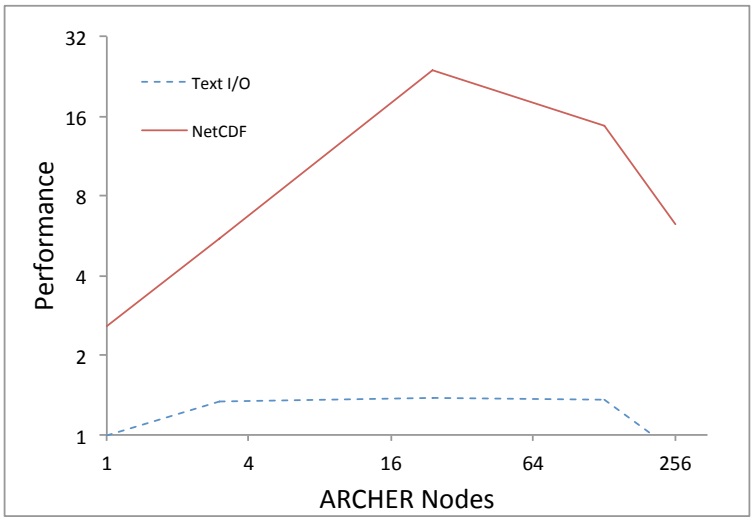


**Figure 3.** Performance of TPLS between 24 and 6144 cores on ARCHER comparing the NetCDF and text I/O implementations

Figure 3 shows the performance of TPLS using both I/O methods up to 6144 processors. The performance metric shown is a speedup relative to the time taken using the original text I/O method on 1 node. Here we demonstrate that when using the original I/O routines, performance is limited to 1.4 on 24 nodes (576 cores), compared to 23.9 using the NetCDF enabled I/O routines. Scaling of the NetCDF I/O is also significantly better, showing a speedup of 9.3 on 24 nodes over the single-node performance. This will greatly facilitate the simulation of larger systems in the future which has currently been

**Table 3.** Time in seconds of NetCDF I/O with different stripe settings

| Number of cores | Number of OSTs 1 | 4 | 8 | 48 |
|---|---|---|---|---|
| 24 | 151.30 | 134.27 | 131.60 | 165.54 |
| 72 | 126.83 | 94.38 | 93.03 | 158.92 |
| 576 | 232.48 | 170.86 | 150.95 | 264.42 |
| 3072 | 401.76 | 243.55 | 227.66 | 348.36 |
| 6144 | 606.76 | 453.25 | 417.56 | 804.07 |

restricted by the I/O overhead, memory bottleneck and the limited control over write frequency.

### *3.2. The effects of using striping in I/O*

A further aspect of investigation in this work was to assess the optimal 'data striping' for use in TPLS. The Lustre parallel file system available on ARCHER is accessed via a set of IO servers — Object Storage Servers (OSSs) — which access data provided by Object Storage Targets (OSTs). A file is 'striped' when it is spread over multiple OSTs and read or write operation on such a file access multiple OSTs concurrently. File striping thus provides a simple way to potentially increase I/O performance in many applications but is often not investigated. Thus for the final part of our work we assessed the optimal striping for TPLS, and whether striping would have a significant impact on performance.

Given the architecture of the ARCHER Lustre parallel file system, the 'single file, collective write' model used in the new NetCDF I/O implementation should benefit from striping each file across multiple OSTs. We compared the performance of the file write operations using the same simulation set up as in Section 3.1, striping the output files across 1, 4, 8 and 48 OSTs per file. By default on ARCHER all files are striped across 4 OSTs, unless otherwise specified by the user and the maximum striping possible is 48 due to the number of OSTs per file system. Lustre stripes files in a block-cyclic fashion with a fixed block size of 1 MiB.

The data presented in section 3.1 used the ARCHER default data striping (4 stripes). Table 3 shows that additional performance improvements (up to 12% on 576 cores) can be gained by striping over 8 OSTs although the effect is small compared to the minimum (1) and the maximum (48) stripes. For simulations using even larger grid sizes, we expect that further improvement may be gained by striping the data across more than 8 OSTs.

## 4. Conclusion

Due to the use of established scientific libraries such as PETSc and NetCDF over bespoke solutions, we have shown TPLS to be a better performing and more scalable code. In addition, refactoring the code structure to remove the need for editing code in order to set up different simulations makes TPLS much more usable and enables it to be provided as a precompiled package on ARCHER.

Two major challenges remain unsolved in the current released version: firstly, to take full advantage of the scalability provided by PETSc, in particular the ability to use a 3D

domain decomposition, requires porting all of the solvers in the code to use this framework. Once this is complete the additional overhead of copying data to/from PETSc data structures at each time step can also be avoided. Secondly, development of additional physics, boundary conditions, and geometries takes place to date in separate branch versions of the TPLS code. To take full advantage of the performance improvements which are implemented in the trunk, these must be merged.

Nevertheless, TPLS usage is already growing, and as well as being used for research applications [1,2] has also attracted the interest of industrial users such as Shell R&D.

## Acknowledgement

This work was funded by the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk) and the HECToR Distributed Computational Science and Engineering (CSE) Service operated by NAG Ltd. HECToR – A Research Councils UK High End Computing Service — was the UK's national supercomputing service, managed by EPSRC on behalf of the participating Research Councils. Its mission was to support capability science and engineering in UK academia. The HECToR supercomputers were managed by UoE HPCx Ltd and the CSE Support Service was provided by NAG Ltd (http://www.hector.ac.uk). Additional support was provided by the EPSRC Grant EP/K00963X/1 "Boiling in Microchannels".

## References

[1] Lennon Ó Náraigh, Prashant Valluri, David M. Scott, Iain Bethune, and Peter D. M. Spelt. Linear instability, nonlinear instability and ligament dynamics in three-dimensional laminar two-layer liquidliquid flows. *Journal of Fluid Mechanics*, 750:464–506, 7 2014.

[2] Patrick Schmidt, Lennon Ó Náraigh, Mathieu Lucquiaud, and Prashant Valluri. Linear and nonlinear instability in vertical counter-current laminar gas-liquid flows. *arXiv preprint arXiv:1507.04504*, 2015.

[3] P. J. Sáenz, K. Sefiane, J. Kim, O. K. Matar, and P. Valluri. Evaporation of sessile drops: a three-dimensional approach. *Journal of Fluid Mechanics*, 772:705–739, 6 2015.

[4] Mark Sussman and Emad Fatemi. An efficient, interface-preserving level set redistancing algorithm and its application to interfacial incompressible fluid flow. *SIAM Journal on scientific computing*, 20(4):1165–1191, 1999.

[5] Hang Ding, Peter DM Spelt, and Chang Shu. Diffuse interface model for incompressible two-phase flows with large density ratios. *Journal of Computational Physics*, 226(2):2078–2095, 2007.

[6] TPLS: High Resolution Direct Numerical Simulation of Two-Phase Flows. http://sourceforge.net/projects/tpls/.

[7] David Scott, Lennon Ó Náraigh, Iain Bethune, Prashant Valluri, and Peter D. M. Spelt. Performance Enhancement and Optimization of the TPLS and DIM Two-Phase Flow Solvers. *HECToR dCSE Report*, 2013.

[8] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, Stefano Zampini, and Hong Zhang. PETSc Web page. http://www.mcs.anl.gov/petsc, 2015.

[9] NetCDF web page. http://www.unidata.ucar.edu/software/netcdf.

[10] The HDF Group. Hierarchical Data Format, version 5. http://www.hdfgroup.org/HDF5/, 1997-2015.

[11] John W. Cahn and John E. Hilliard. Free energy of a nonuniform system. i. interfacial free energy. *The Journal of Chemical Physics*, 28(2):258–267, 1958.

[12] Gavin Pringle. Porting OpenFOAM to HECToR. *HECToR dCSE Report*, 2010.

[13] Greg Clifford and Scott Suchyta. Staying Out of the Wind Tunnel with Virtual Aerodynamics. *Proceedings of the Cray User Group (CUG), 2015*, 2015.