ARCHER ECSE 05-14 LARGE-EDDY SIMULATION CODE FOR CITY SCALE ENVIRONMENTS

TECHNICAL REPORT

Vladimír Fuka, Zheng-Tong Xie

Abstract

The atmospheric large eddy simulation code ELMM (Extended Large-eddy Microscale Model) is optimized and its capabilities are enhanced to allow simulations of large domains in high resolution in the areas of main interest. A hybrid OpenMP/MPI version of the code is developed. The new features developed include one-way and two-way grid nesting. This enables simulation of large domain in affordable resolution and high resolution in selected areas and still maintaining a simple uniform grid with the associated code simplicity and efficiency.

Description of the software

The package ELMM (Fuka, 2015; https://bitbucket.org/LadaF/elmm) solves turbulent flow in the atmosphere optionally with the effects of temperature and moisture stratification. The filtered incompressible Navier-Stokes equations are discretized using the finite volume method. The effects of subgrid turbulence are modelled using several alternative subgrid models.

The pressure-velocity coupling is achieved using the projection method. In this method a discrete Poisson equation is solved. The solution of this elliptic system typically requires a considerable part of the total CPU time. The Poisson equation is solved using the custom library PoisFFT (Fuka, 2015; https://github.com/LadaF/PoisFFT) which uses the discrete fast Fourier transform from the FFTW3 library (Frigo and Johnson, 2005). The PFFT library (Pippig, 2013) is used for distributed fast Fourier transforms on top of FFTW3.

At the start of the project ELMM and PoisFFT could be compiled either with OpenMP or with MPI parallelization, but not with a hybrid combination of both.

The usage of FFT is possible because of the uniform orthogonal grid used by ELMM. This however means that complex geometries must be modelled using the immersed boundary method and the grid resolution is the same in all parts of the grid.

The Work Packages

The work plan for eCSE 05-14 was divided into 2 work packages, each consisting of 2 subpackages.

WP 1 Optimisation of parallelisation

Making ELMM run faster for computations it is currently capable of and enabling larger scale problems than those currently possible.

WP 1.1 Hybrid parallelisation

Optimization of the serial code based on profiling. Optimization of current OpenMP and MPI parallelization. Combining MPI and OpenMP into a hybrid approach.

WP 1.2 Parallel I/O

Parallel MPI based input-output of data. Currently each ELMM process stores its part of the data in a separate file in its own subdirectory. Parallel I/O changes this into storing of one large file reducing the load of the file-system.

WP 2 Grid nesting

Also called domain nesting. Developing a new capability which enables simulation of large areas with a coarse grid and at the same time simulation of selected areas in high resolution.

WP 2.1 One-way nesting

In one-way nesting approach, the nested grid domain can behave like a usual model run, but must get its boundary conditions from the host domain. The boundary conditions must be interpolated from the processes which constitute the outer coarse domain. The outer domain does not influence the flow in the outer domain.

WP 2.2 Two-way nesting

In two-way nesting the inner domain solution feedbacks on the outer domain solution.

Results of WP1.1

The serial efficiency of ELMM was improved in several ways. Several important 3D loop nests contained `if` conditions which tested if the grid point needs to be treated differently due to closeness of a solid wall. These were eliminated. In certain cases this means duplicating of part of the work as certain terms have to be first added in all points and then subtracted later when the point is found in a list of wall points. An example would be too large to fit in this report. The reader can refer to the source code repository at https://bitbucket.org/LadaF/elmm/.../src/dynamics.f90 and compare subroutines `MomentumDiffusion` and `MomentumDiffusion_nobranch` (the old and the new version). The overall effect is clearly positive. A single node comparison of ELMM showed 20% speed increase after optimization.

To optimize OpenMP parallelized loops the collapse clause was added where applicable to extend the potential for scaling when the number indexes in the outermost loop is small. Extensive testing was used to determine best scheduling parameters. Different loop nests use static, dynamic and guided scheduling based on the testing. The most important 3D loops have run-time scheduling and the final scheduling type can be set by the user based on tests made for a particular problem on a particular machine. Our tests on ARCHER suggest that the dynamic scheduling is optimal in most cases.

Hybrid OpenMP/MPI parallelism was developed in ELMM assuming the MPI_THREAD_SERIALIZED threading support in the MPI library. That means that no 2 threads are permitted to execute MPI communication at the same time in a single MPI process. This choice was made because of the low support of MPI_THREAD_MULTIPLE in high performance libraries.

The most difficult part of the update to hybrid parallelism was the PoisFFT library. Depending on the boundary conditions either the 3D distributed fast Fourier transforms routines from the PFFT library are called or custom scheme calling a sequence of lower-dimensional transposes is called. The distributed fast Fourier transforms require several steps of global array transpose for reordering the array so that the second or third array index is laid out contiguously in memory of each process and distributed between the processes in the right way. In PFFT this step is done automatically, but for the custom scheme it is done in PoisFFT explicitly. In this important step it was only possible only to parallelize by threads the local re-ordering operations while the call to MPI_AllToAllV() is only performed by a single thread. It was expected that the hybrid parallelism will help with the scaling of the Poisson solver for high numbers of cores by lowering the number of processes to communicate in MPI_AllToAllV() by an order of magnitude.

The solution of the Poisson equation is the part which is expected to have the most difficulties with the parallel scaling because the equation is elliptic and the solution in every point is dependent on every other grid points distributed across all processes.

SCALING

The parallel scaling of ELMM was tested on ARCHER with the GNU (GCC) programming environment version 5.2, FFTW version 3.3.4 and PFFT library version 1.0.6.

The test-case selected was derived from the geometry used in the DIPLOS project. The grids used routinely in the project used 7.1 and 28.3 millions of cells. The boundary conditions are a solid wall on the bottom and free-slip boundaries on the top and in the spanwise direction. The inflow has a synthetic generator of turbulence and the outflow is a zero-gradient condition with a sponge layer damping any incoming fluctuations. In the weak scaling test the grid size for one node was kept constant at 885 thousand cells. In the strong scaling test the whole domain size was kept constant at 56.6 million cells.

The total wall times of the program were divided into two parts. The time spent in the Poisson solver and the remaining part spent in ELMM proper.

We tested both the strong scaling and the weak scaling and the pure MPI version and the hybrid OpenMP/MPI version.

Because the Poisson solver PoisFFT uses the fast Fourier transform (FFT) the complexity of the algorithm does not grow linearly with the size of the problem but with O(n*log(n)). Therefore, linear scaling of the Poisson solver cannot be expected.

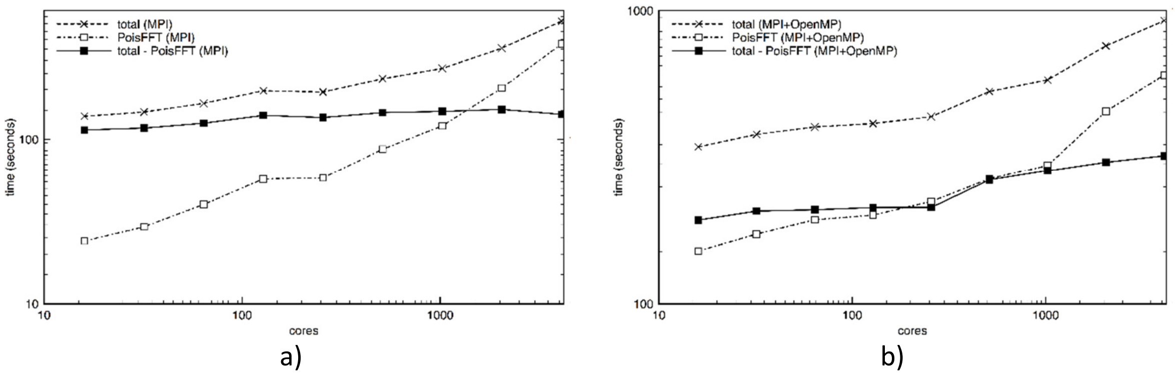

Figure 1 Weak scaling of ELMM performance: a) pure MPI, b) hybrid MPI/OpenMP

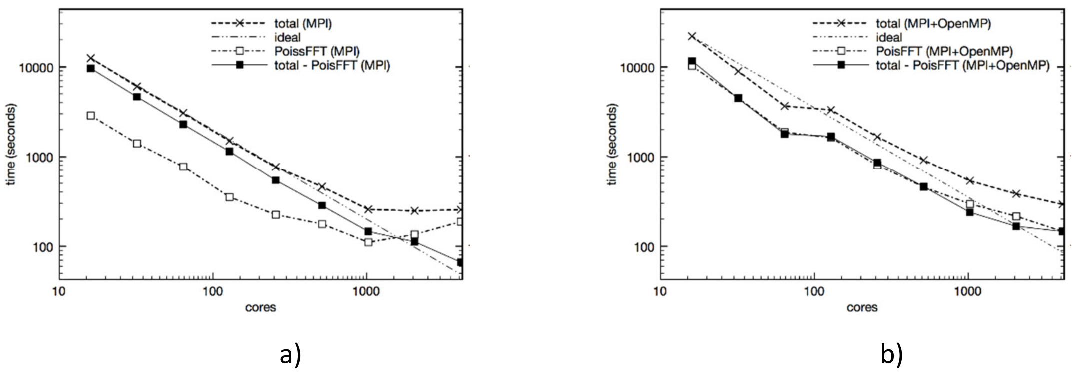

Figure 2 Strong scaling of ELMM performance: a) pure MPI, b) hybrid MPI/OpenMP

The first point one can immediately notice in Figures 1 and 2 is that the hybrid version of the Poisson solver is much slower than the pure MPI one for small number of nodes. For these boundary conditions the whole FFT work is done inside a single call to the distributed 3D FFT routine from the PFFT library. The authors of this library was able to demonstrate good weak scaling of the pure MPI version PFFT up to 65 thousands of cores. We were not able to replicate this result on ARCHER and PFFT stops to scale well sooner in our configuration.

One can also see that the Poisson solver always becomes the dominant part of the whole solution time at certain number of CPU cores. The ELMM without the Poisson solver scales better than the Poisson solver. The weak scaling of this part is almost flat for the pure MPI version (Fig. 1a, full line).

In Fig. 2a) one can see that the pure MPI version of PoisFFT stops to scale completely at 1024 cores while the hybrid version wall-clock time still decreases at 4096 cores where it becomes faster than the MPI version. In the weak scaling test the hybrid version of PoisFFT starts as being much slower than the pure MPI version, but the solution time grows more slowly with increasing the number of the CPU cores.

It is expected the hybrid parallelism will be even more extensively employed when new types of accelerators coming to the largest supercomputers will be used for computations. For present type of computer nodes used in ARCHER the overhead of using threads in addition to MPI and the additional synchronizations associated with that are expected to slower the code in comparison with the pure MPI approach for moderate numbers of nodes.

Results of WP1.2

The attempt of the assigned EPCC staff to develop the parallel I/O feature was unsuccessful. The number of the files can still be decreased by a factor of 12 or 24 when using the hybrid parallelization which uses a smaller number of MPI processes.

Results of WP2.1

The one-way nesting procedure required preparation of a new Fortran module which collected the information about the grids of all other domains and individual processes of those domains. After introduction of this module every process has the following data about each domain: the dimensions in each direction, number of cells in each direction and their coordinates, whether that domain is an inner or an outer domain for the current domain. Similar data are also collected about individual processes within each domain and in addition to them also the MPI ranks of each remote process.

This gathering of information is necessary, because the domains can be organized freely in a hierarchical tree. The original outer boundary can contain one or more inner domains. Each of these inner domains can have one or more inner domains themselves.

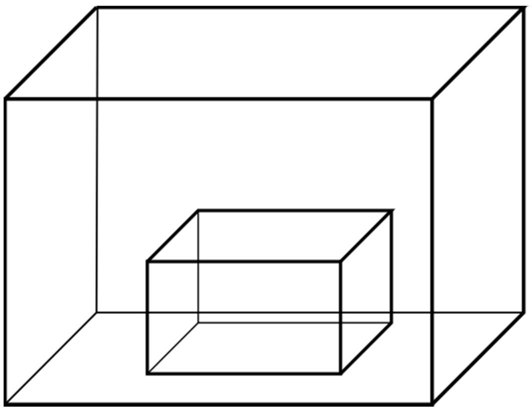

Figure 3 An outer domain and a nested inner domain.

An important limitation, made to greatly simplify the coding, is that each process in any inner domain must have just one parent process in the outer domain. This poses a limitation on how we can divide the domains into individual processes and limits the possible locations for inner domain boundaries somewhat. The hybrid parallelism developed in WP1.1 brings some additional freedom because the same number of cores can be utilized with various numbers of MPI processes depending on the number of OpenMP threads per process.

Another limitation, but common in nesting approaches, is that every cell of any outer domain must be equally divided into a fixed number of cells in the inner domain. That also implies that the boundary of an inner domain must lie on a boundary between cells of the outer domain. Currently the grid refinement ratio must be equal in all three directions. The ratio is a positive integer number chosen by the user in the configuration file and can differ between individual inner domains.

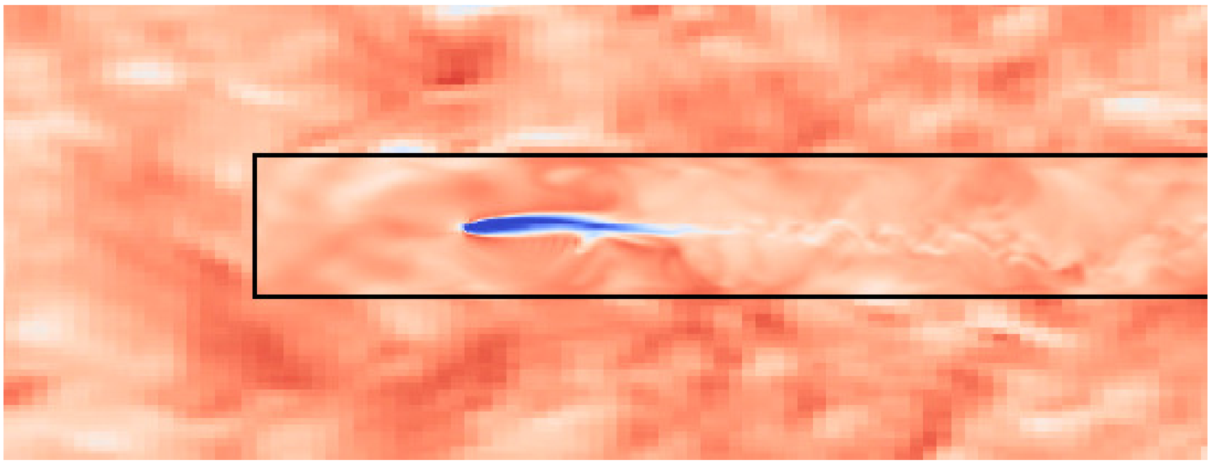

Figure 4 A flow around an airfoil. The outer domain outside enables a turbulent inflow field with a small blockage and the inner domain brings the required resolution near the object.

Each inner domain can have one or more nesting boundary conditions and classical boundary conditions in the domain boundary. On nesting boundaries the boundary conditions are determined from the flow variables and their tendencies from the last time-stop on the outer domain. The fields are interpolated on the denser grid. The tendencies are used to extrapolate the fields in time.

Optionally, the boundary surface itself is accompanied by another layer of cells (buffers) where nudging is used to force the fields inside the inner domain, but close to the boundary, to become close to the outer domain values at that position. It can be viewed as a force acting in the inner domain close to the boundaries always forcing the inner flow to match the outer flow.

It is possible to prepare an inner domain which only has one nesting boundary at inflow. That way one can use the nesting capability for generating turbulent inflow by the recycling method.

It was found out that the coarser flow field interpolated on the inner domain boundaries is missing small scales in the flow which are not resolved by the outer grid, but should be represented by the inner grid. These scales can develop themselves by the turbulent decay of the larger scales but our tests revealed this can take too much time and can affect the inner domain solution relatively far from the boundary. We developed a method in which we employ the synthetic turbulence generator (Xie and Castro, 2008), traditionally used to generate the complete synthetic flow field, to generate the missing small scales. This method was successfully tested on a plane channel flow (Fig. 5).

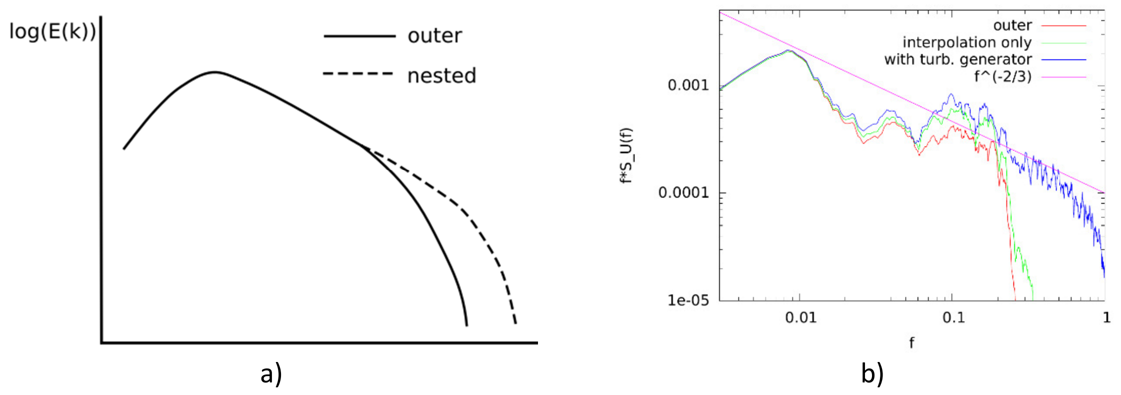

Figure 5 Temporal spectra of turbulent fluctuations in grid nesting. a) Theoretical spectra of two LES simulations with different grid resolutions. b) Temporal spectrum of the u velocity component close to the inflow nesting boundary in a plane channel with grid refinement ratio 1:5. Red line – the outer domain spectrum, greed line – inner domain spectrum if only interpolation from the outer domain is used, blue line – interpolation and a synthetic generator of the small scales of turbulence, magenta line – theoretical scaling in the inertial range.

Results of WP2.2

The two-way nesting procedure was developed to enable the feedback of the high resolution solution from the inner domain on the outer grid. It was implemented in the following way:

1. At the end of the predictor step of the momentum equation in the nth time-step of the inner domain and in every time-step of the outer domain the inner domain fields are filtered onto the grid of the outer domain and sent to the outer domain.
2. The outer domain will copy the values of the filtered field into its grid and replace the current values. It replaces the values only inside the fluid, i.e. only outside of obstacles and outside of the regions which serve as a source of the boundary nudging buffers in the inner domain (see the description of the one-way nesting above).
3. New boundary conditions are computed based on the new fields' values.
4. The Poisson equation is solved with the divergence of the current velocity field to project the velocity field to a non-divergent velocity field. New pressure field is computed in this step.

The importance and impact of using two-way nesting and the implications of it to the accuracy of LES simulations is still an open research problem. An illustration of its usefulness is the use of nested grids which include a scalar source. The nested grid allows to resolve the scalar source and the dispersion in its vicinity in high resolution and the large outer grid allows to simulate the dispersion of the scalar to a large distance. The example in Fig. 6 is computed using three nested domains with grid refinement ratio 2x. The use of two-way nesting allowed to perform this simulation on a 4-core CPU workstation in a few hours. A domain of the horizontal size of the present middle domain and vertical size of the present outer domain already required 24 ARCHER nodes for our previous simulations in high resolution used for the current innermost domain.

Application of two-way nesting in large eddy simulation is still an area of active research and there are still issues to be solved. For example, we identified that the interpolation from the outer grid to the inner grid will reduce the turbulent kinetic energy on the inner grid between the outer grid points. This effect is explained and quantified by Veers (1988), although not in the context of grid nesting and LES. The missing turbulent kinetic energy can then also effect the outer domain through the two-way nesting. It could be added back using the synthetic turbulence generator, but the amount required varies strongly from point to point and depends on quantities not easily available, like point-to-point correlations. This will be one of the directions of future research.

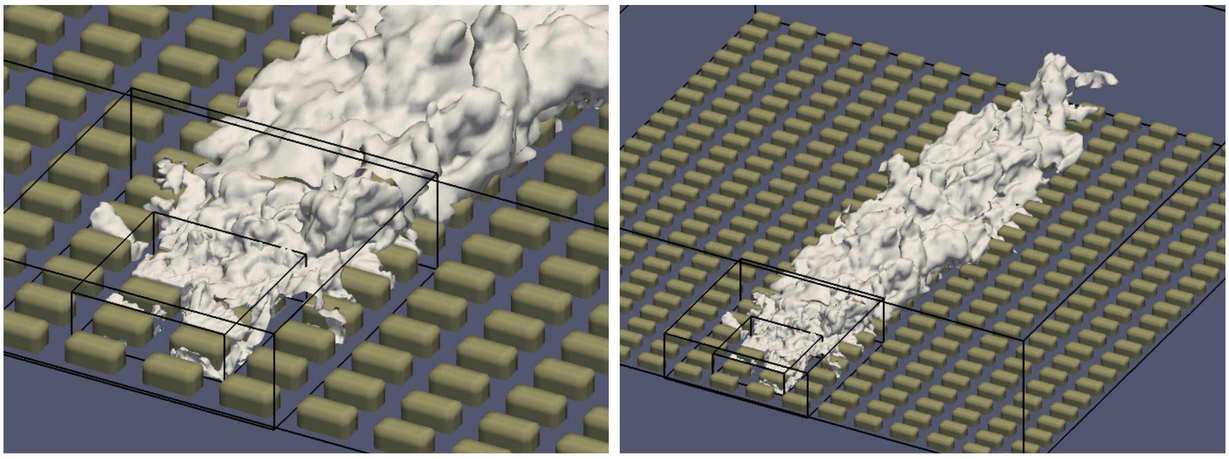
a) b)

Figure 6 Instantaneous concentration isocontours from point source dispersion with the source located within the urban canopy layer. This geometry is used within the DIPLOS project (EPSRC). Two-way nesting of scalar concentration was used. The source injects the scalar to the innermost domain only, the outer domains receive the scalar concentration by the two-way nesting. Domain size 48 *h* x 48 *h* (*h* is the building height). Domain resolutions 4 cells/ *h*, 8 cells/*h* and 16 cells/*h*. a) Detail of the nested domains, different smoothness of the isocontour in different levels of nesting is apparent. b) The complete plume.

CONCLUSION

The LES code ELMM has been improved in several ways within the project. The single node performance was improved by several optimizations in the most intensive loops. The MPI data exchange between neighbouring processes was translated to non-blocking MPI communication and the number of necessary synchronizations reduced. The hybrid OpenMP/MPI parallelization was introduced. It enables higher flexibility in choosing the grid distribution between processes, but is slower for small numbers of CPUs. This difference diminishes for large CPU counts (Fig. 2).

One-way and two-way nesting were developed as new features of the code. This brings the possibility to simulate very large domains in affordable resolution and smaller domains in high resolution. The configuration is very flexible and can be also used for creating recycling inflow boundary conditions and similar configurations. The synthetic turbulence generator has been demonstrated to be effective for producing small-scale turbulence in the inner domain that is not resolved on the outer grid.

The code has been demonstrated to simulate domains much larger than the neighbourhood scale (~ 1km) and has a potential to simulate domains of size ~10 km.

REFERENCES

V. Fuka, PoisFFT – A free parallel fast Poisson solver, Applied Mathematics and Computation 267 (2015) 356–364, http://dx.doi.org/10.1016/j.amc.2015.03.011.

V. Fuka, Z.-T. Xie, Domain nesting for multi-scale large eddy simulation, European Geophysical Union General Assembly, 21.4. 2016.

M. Frigo, S. Johnson, The design and implementation of FFTW3, Proc. IEEE, 0018-9219 93 (2) (2005) 216–231. ISSN 0018–9219.

M. Pippig, PFFT: an extension of FFTW to massively parallel architectures, SIAMJSC 35 (03) (2013) C213–C236, http://dx.doi.org/10.1109/JPROC.2004.840301.

Z.-T. Xie, I. P. Castro, Efficient generation of inflow conditions for large eddy simulation of street-scale flows, Flow, turbulence and combustion 81.3 (2008) 449-470.

P. S. Veers, Three-Dimensional Wind Simulations, Sandia National Laboratories, SAND88-0152 (1988), http://prod.sandia.gov/techlib/access-control.cgi/1988/880152.pdf.