The aerodynamics of helicopter rotor blades is one of the most interesting and challenging problems facing aerodynamicists. Predicting the flow around a blade is a totally different problem to an aircraft wing, which has an (almost) uniform constant speed flow over it, and what happens downstream of the wing is not that significant. That is not the case for a rotor blade, where the speed of the blade varies from very low at the root, to very high at the tip, and in forward flight the effective velocity the blade sees is different at every point around the azimuth (revolution). Hence, the flow is highly three-dimensional and unsteady. Furthermore, and most significantly, each blade moves into fluid that has already been disturbed by the previous blade(s). The loud buzzing, vibrating, sound caused by helicopters is due to the wake, and particularly the strong vortex from each blade tip, shed from one blade being hit by the following one. The accurate prediction of this blade-vortex interaction (BVI) is essential for both civil helicopters, to attempt to reduce ‘noise pollution’, and military, to avoid detection.

Until recently, numerical simulation of these flows was rare, due to excessive cost, but rapidly increasing computer power and code capability have meant this is now possible.

Computational methods for fluid flow simulation involve filling the physical domain of interest with a computational grid, ie filling the domain with a number of cells, each one having the solution stored in it. The solution is the local values of flow variables, ie density, pressure, velocity, etc. Using applied techniques from mathematics and physics, methods can be developed which march the solution forward in time, starting from an initial guess and iterating until the solution converges (stops changing), and these methods can be used to simulate steady or unsteady flows. The numerical approximations have a truncation error associated with them, ie the difference between the real equations and the numerical approximation, and the error is a function of the grid spacing. Hence, the finer the computational grid used, the more accurate the solution is. The error in the numerical approximation is known as dissipation, and so the coarser the grid the more dissipation is added to the flow and, hence, the interesting parts of the flow are ‘smeared out’.

This leads to the major problem with simulation of rotor flows. The aerodynamicist’s job is to predict forces on wings or blades, and to do this the flow on the surface is required. For a fixed-wing case, grid points can be clustered close to the surface so the flow is computed accurately there, and the grid away from the surface can be coarse as the flow there is of little significance. However, as
mentioned above, the flow around a rotor is significantly affected by the wake from previous blades. Hence, to simulate rotor flows requires extremely fine meshes throughout the grid, otherwise the wake from each blade is dissipated by the numerical scheme before it hits the next one. Furthermore, to capture the wake over many turns, particularly for hovering rotors where a helical wake develops beneath the blades, requires a large number of iterations/time-steps. Hence, rotor flow simulation requires many time-steps on a very fine mesh, and this leads to huge run times.

Flow-solver and aspects of parallelisation
Dr Allen has developed both flow-solver and grid generation methods for rotary wing applications. The code is a structured multiblock, upwind solver, using implicit time-stepping for unsteady calculations, with explicit-type time-stepping within each real time step. Multigrid acceleration is used to improve convergence. The multiblock approach adopted is ideal for parallelisation, and the code has been parallelised using MPI. The code has been written so that for each grid block it only requires the IMAX, JMAX, KMAX dimensions, the IMAX×JMAX×KMAX coordinates, and one line per block face listing the boundary condition flags. Hence, each block can be written to a separate file, and a header file lists the name of each block file. All connectivity
data and multigrid data is computed and stored locally only. At any internal block boundary, i.e., connected to another block, the two adjacent planes of solution data are simply packed into an array and sent to the processor on which the neighbour block is stored, and vice-versa. Any incoming data is then unpacked according to an orientation flag. This allows all sends/receives to be non-blocking, and also makes future extensions straightforward, for example moving to a higher-order stencil.

To ensure maximum efficiency and, hence, grid sizes, there is no global data storage, so each processor only needs to store geometric and flow solution data for its blocks. This is possible due to the multiblock nature, which allows partitioning to be done separately. To this end a preprocessor has been developed, which scans the grid block sizes and maximises the load balance while also attempting to maintain the maximum number of multigrid levels.

Grid generation is also significant here. It is essential that the possible solution grid size is not limited by the grid generator and, hence, a multiblock generation tool has been developed that can generate a 1000 block, 64 million point rotor mesh in around 30 CPU minutes on a P4 Linux machine, requiring less than 2 GBytes RAM.
Results and parallel performance

A wake grid dependence study was performed for a four-bladed rotor in forward flight. The computational domain, and block structure was kept constant and grids of size 1, 2, 4, 8, 16, and 32 million points were generated. Figure 1 shows the computational domain and block boundaries. The blade is the ONERA 7A rectangular blade, aspect ratio 15, ie tip radius is 15 blade chords, and the domain is a cylinder radius 50 chords, height 80 chords. There are 408 blocks. Figure 2 shows the grid in the rotor disk for 1, 8, and 32 million points.

The case chosen was a shallow descending case. The tip Mach number is 0.618, the advance ratio is 0.214 (so forward Mach number is 0.1322), and the rotor shaft is inclined 3.72 degrees backward. Hence, this has significant BVI effects. The results were computed on all six grids, using 180 real time-steps per revolution, and three revolutions were computed to obtain periodicity.

Figure 3 shows vorticity shading on selected grid planes, for the 1, 8, and 32 million point grids (the scales are the same in each). The effect of numerical dissipation is clear. The interesting consideration here is the grid density effect on local and global quantities. Figure 4 shows the total load coefficient of the lead blade around the azimuth. Hence, it is clear that if total load is of interest, then 32 million points are not required. However, figure 5 shows the local normal force coefficient, for blade sections at 50% and 82% of tip radius respectively, around the azimuth. The peaks are caused by BVI effects and, hence, it seems that grid convergence has not been achieved even with 32 million points.

As stated earlier, these calculations are extremely expensive. The 16 million point case was run on 256 CPUs on HPCx, and 32 million point case on 512 CPUs. The code has also been subjected to scaling tests, using a 20 million point mesh, and figure 6 shows the parallel performance (assuming a speed-up of 1 with 32 CPU’s). Hence, excellent scaling has been achieved, and the code has been awarded a gold star for performance.

Acknowledgements

The HPCx computer time was provided through the UK Applied Aerodynamics Consortium (of which the author is a member) under EPSRC grant EP/S91130/01. Thanks are also due to Andy Sunderland, of the Terascalcing Team, for performing the scaling tests.