European Conference on Computational Fluid Dynamics
ECCOMAS CFD 2006
P. Wesseling, E. Oñate and J. Périaux (Eds)

PARALLELISATION OF CFD METHODS FOR MULTI-PHYSICS PROBLEMS

René Steijl*, George N. Barakos* and Ken J. Badcock*

*University of Liverpool, Department of Engineering
Brownlow Hill, Liverpool L69 3GH Liverpool, United Kingdom
e-mail: R.Steijl@Liverpool.ac.uk

Key words: CFD, Multi-disciplinary problems, parallel computing

Abstract. *The focus of this paper is on the parallel computing aspects of multi-physics CFD methods and the implications for the design of a CFD framework encompassing mesh-based and particle-based approaches. In the first part of the paper, the shared-memory parallelisation of a multi-block, structured CFD is described and the parallel performance of this method is assessed for multi-block meshes for realistic complex configurations. It was found that for the coarse-grained approach taken, a limited parallel scaling was obtained, indicating that a hybrid shared-memory/message-passing approach will be of particular interest. The second part of the paper describes the design and parallelisation of an object-oriented molecular dynamics method, which will form a particle-based component of the multi-physics CFD framework. It is shown that for the object-oriented implementation of this method, the shared-memory paradigm leads to an excellent parallel performance.*

1 INTRODUCTION

Aerodynamic flows of practical interest always present multi-disciplinary problems. Aero-elasticity, control of boundary layers, flows around aircraft in iced conditions and re-entry flows are some good examples which highlight the multiple levels of physics present in applied aerodynamics. As Computational Fluid Dynamics is maturing, emphasis is shifted towards accurate prediction of such flows and consequently the development of methods which include a number of physical models with corresponding sets of governing equations and numerical solution techniques. Such methods are usually termed multi-physics CFD methods and are the focus of this work. This work describes the initial steps in the development of a framework for multi-physics CFD, based on mesh-based as well as particle-based approaches.

The computation of 3D unsteady flow fields using CFD is a highly demanding computational task and the development of parallel CFD methods has been an active field of research for more than a decade. For finite-volume CFD solvers, the domain-decomposition approach using message-passing for exchanging data between flow solutions in the sub-

domains and the synchronisation of the parallel processes is now considered to be standard. Using libraries like MPI, the message-passing approach leads to portable methods with good parallel efficiency on distributed-memory (including Linux clusters) as well as shared-memory machines. Shared-memory multiprocessors can be divided in two main categories, symmetric multiprocessors when the cost of a memory access is the same no matter which CPU (or thread) performs this operation and typically have a small number of CPUs, and machines with Non-Uniform Memory Access (NUMA). For shared-memory machines, the OpenMP standard provides an application programming interface (API) for shared-memory parallelism. However, shared-memory parallelisation has seen a limited application in CFD. Regardless, the OpenMP standard has been successfully used by some research groups for the the development of CFD solvers with reasonable parallel efficiencies for up to 16 processors[1, 2]. For multi-physics, inter-disciplinary CFD methods, which combine various physical models and numerical solution methods for the flow field, possibly coupled with structural, acoustic or flow-control models, parallelisation based on the message-passing paradigm may not be the most efficient paradigm. For such CFD methods, shared-memory parallelism for the coupled problem or for one or more of the different numerical methods involved, may be a more natural candidate. This consideration forms the main motivation for the present investigation in shared-memory parallelisation of inter-disciplinary CFD methods. Addition motivation stems from recent trends in computer hardware development, e.g. the introduction of affordable CPU chips with multiple cores, and the current trend to build modern high-end super-computers on a shared/distributed memory architecture, i.e. as clusters of shared-memory multi-processor nodes.

The present work can be regarded as the first steps in an a research effort into the development of efficient parallel multi-physics CFD methods, where the OpenMP standard forms the basis for shared-memory parallelism, for finite-volume methods as well as particle-based methods. The long-term aim of the present work is the formulation and development of a framework that unifies mesh-based, finite volume CFD methods with particle methods.

Section 2 of this paper describes the development of a shared-memory parallelisation of a multi-block, structured CFD solver[3, 4]. The parallelisation, the parallel performance and a comparison with the MPI implementation are discussed in detail. A next step in this investigation is the assessment of the hybrid MPI/OpenMP paradigm, which employs shared-memory parallelism within and message passing between the shared-memory multi-processor nodes. This first step enables an assessment of the merits of shared-memory parallelisation of multi-block, structured CFD solvers applied to realistic, complex configurations. Section 3 describes the development of a parallel object-oriented Molecular Dynamics method, highlighting the parallelisation of particle-based solution methods. Section 4 of this paper summarises the findings of the ongoing work on parallel mesh-based CFD methods and the development of parallel particle-based methods and discusses the future directions.

2 BLOCK-STRUCTURED FINITE-VOLUME CFD SOLVER

The starting point of the investigation is a multi-block, structured solver which has been used previously for a wide range of applications[3, 4]. The parallelisation of this method is based on the domain-decomposition approach along with the message-passing paradigm for communication and synchronisation of the solution in the sub-domains. In order to gain understanding of the merits of shared-memory parallelisation, a version of the solver was developed based on the OpenMP standard.

2.1 Governing equations and discretisation method

The unsteady Navier-Stokes equations are discretised on a curvilinear, multi-block, body conforming mesh using a cell-centred finite volume method. The convective terms are discretised using Osher's upwind scheme [5] and MUSCL variable extrapolation is used to provide nominally third-order accuracy on a uniform mesh. The Van Albada limiter is used to prevent spurious oscillations around shocks. Central differences are used for the discretisation of the viscous fluxes. The solver includes a range of one- and two-equation turbulence models as well as LES (based on the Smagorinsky sub-grid model) and DES (based on the Spalart-Almaras and $k-\omega$ models). Dual-time stepping is employed for time-accurate simulations, where the time derivative is approximated by second-order backward differences[6], while the resulting non-linear system of equations is solved by integration in pseudo-time using first-order backward differences. In each pseudo-time step, a linearisation in pseudo-time is used to obtain a linear system of equations, which is solved using a Generalised Conjugate Gradient method with a Block Incomplete Lower-Upper (BILU) pre-conditioner. The method is detailed in Ref.[3]. The flux Jacobians resulting from the linearisation in pseudo-time are employed in an approximate form that reduces the number of non-zero entries and the size of the linear system. The use of approximate Jacobians also reduces the parallel communication since only one row of halo cells is needed across block boundaries in the linear solver instead of two in the case of an 'exact' Jacobian.

2.2 Implementation

The CFD solver considered here[3, 4] is written in C and uses a data structure based on dynamically allocated arrays and linked-lists. In the shared-memory paradigm, loops and/or program blocks ("parallel regions") of a program were selected for parallel execution. This selection was based on the data inter-dependence within the loop or program block. For the selected loops and parallel regions, compiler directives were inserted to create threads that run in parallel to the main serial part of the program. The overhead associated with thread creation, destruction and synchronisation is significant, and consequently the selected loops should involve a significant amount of computation. Other important issues in shared-memory parallelisation are consistent thread scheduling[1], the choice of static or dynamic loop scheduling, cache utilisation, dynamic memory allocation

within parallelised loops and the use of pointer operations. The multi-block formulation of the CFD solver involves a large number of loops (or linked-list traversals) over the sub-domains, e.g. in the numerical flux computation, construction of the Block Implicit Lower Upper preconditioner and the computation of the matrix-vector multiplications in the GCG method. In the OpenMP implementation, the loops over the sub-domains were selected for parallel execution on multiple threads, which can be regarded as a *coarse-grain* approach.

2.3 Load balancing for MPI and OpenMP implementations

An important aspect of most applications of the present CFD solver is the geometric complexity of the considered problems. The block-structured mesh approach typically leads to the use of grids with topologies of significant complexity. These topologies, including the (minimum) number of blocks, their relative magnitudes and connectivity, are largely determined by the complexity of the considered geometry. An efficient parallel simulation requires a well-balanced distribution of the computational load over the processors, i.e. the total number of grid points of the blocks assigned to each processors should be very similar. Furthermore, the communication overhead needs to be minimised. For the situation where each processor computes multiple sub-domains, this means that for each sub-domain as many neighbour blocks as possible should be allocated on the same processor. However, these parallel efficiency requirements are in most practical cases difficult to realise for complex geometries. Therefore, for this type of geometries, the load balancing and the communication overhead are typically worse than for grids of similar dimensions for more generic geometries.

Here, load balancing effects are investigated using three different test cases. The first two examples are examples of flow simulations about a complex geometry, in this case the flow field around a hovering helicopter rotor. The Caradonna-Tung[7] rotor is a two-bladed model rotor with straight, low-aspect ratio blades with a NACA0012 section. The geometry is sketched in Figure 1. The second rotor test case is the ONERA 7AD1 model rotor in hover with parabolic tips, featuring anhedral, blade twist and an aspect ratio of 15. The geometry is sketched in Figure 2. The multi-block topology used for the present computations is shown in Figure 3. As can be seen, accurate representation of the blade tip and rotor hub regions significantly contributes to the complexity of the blocking, which has a total of 74 grid blocks for the Caradonna-Tung rotor. The more complicated 7AD blade geometry requires additional block sub-divisions in the blade radial direction, leading to a total of 98 blocks for this rotor.

Figure 4 shows the wake visualisation (vorticity magnitude) for the Caradonna-Tung rotor from a computation on a fine grid of 4.5 Million grid points.

The third example is the unsteady transonic flow through a square cavity, as a model for an aircraft weapon bay. In this case, the geometry is sufficiently simple to allow a block decomposition with an optimum number of equally-sized grid blocks. Figure 5 shows the multi-block topology that was used for LES simulations of this test case, while Figure 6

shows a representative result for a free-stream Mach number of 0.85 for the considered cavity aspect ratio and depth.

The load balancing approach used in the MPI implementation consists of two steps: first, the blocks are ordered in order of size, then, the blocks are distributed in a round-robin fashion, with the least loaded processor receiving an extra block. This approach leads to good load balancing in terms of the number of grid points per processor. However, this simple approach does not minimise the communication overhead in the message-passing.

The load distribution of the multi-block Caradonna-Tung mesh is shown in Figure 7 for the MPI implementation, where the grid blocks are colour-coded according to the processor they reside during computations. The colour-coding clearly shows that the current load balancing method does not minimise the communication between the blocks.

Table 1 presents the load (im)balance for the MPI implementation for the two example rotorcraft applications considered here as a function of the number of processors used. The grids used here are relatively coarse, leading to a load imbalance for 16 or more processors. In contrast, the transonic cavity test case, which 256 equally-sized grid blocks leads to a perfect load balancing for up to 256 processors when the used number of processors is a power of two.

The OpenMP standard provides the *schedule* clause to control the load distribution in the parallel loops. A *chunk* size can be specified along with one of the following options: i) *Static*, where iterations in the loop are divided into pieces of size *chunk*. These chunks of the loop are then statically assigned to each thread in a round-robin fashion in the order of the thread number. The chunk size is chosen so that each thread gets one contiguous part of the loop, ii) *Dynamic*, where iterations are divided into pieces of size *chunk* As each thread finishes a part of the iteration space, it dynamically obtains the next set of iterations. As a default, the chunk size is chosen to be 1. iii) *Guided*, where the chunk size is reduced in an exponentially decreasing manner with each completed part of the iteration space. In the method, *chunk* specifies the smallest piece (default is 1). It should be noted that the *dynamic* and *guided* options introduce a significant run-time overhead. Typically, these options are used when the *dynamic* options clearly lead to an unacceptable load imbalance.

Table 2 presents the load (im)balance for the coarse-grained OpenMP implementation. Three different loop scheduling options are shown in this table. Clearly, the default static loop scheduling method would lead to a severe load imbalance for multi-block grids for realistic applications. Depending on the mesh characteristics, the most suitable loop scheduling option needs to be chosen. The loop schedule option can be either hardwired in the compiler directives inserted into the source code or as a parameter to be set as an environment variable, providing more flexibility.

Table 1: Load balancing for MPI-implementation for different applications.

Grid	7AD	C-T	Cavity
blocks	98	74	256
grid points	600,000	1,200,000	8,000,000
nprocs.	imbalance		
2	0.0%	0.0%	0.0%
4	1.1%	0.6%	0.0%
8	1.6%	3.0%	0.0%
16	2.9%	6.8%	0.0%
32	9.9%	13.8%	0.0%

Table 2: Load balancing for different OpenMP loop schedule options for 7AD rotor problem.

	static(default)	dynamic(2) (estimate)	dynamic(4) (estimate)
nprocs.	imbalance		
2	21.0%	7.77%	0.00%
4	30.0%	12.7%	13.0%
6	41.7%	21.3%	28.3%
8	62.0%	23.6%	44.1%

2.4 Parallel performance

The parallel efficiency of the OpenMP implementation was assessed during numerical experiments conducted on two parallel computers at the Bundeshöchstleistungrechenzentrum Stuttgart (HLRS): a 4-way Opteron Symmetric Multi Processor and a NEC TX-7 with 16 Itanium II processors and a cache-coherent NUMA architecture.

The parallel speed-up for the MPI and OpenMP implementations is presented in Figures 8 and 9, respectively. The MPI implementation gives nearly linear speed-up to 16 processors, while the OpenMP implementation produces good speed-up for up to 4 processors. The load imbalance and the overhead associated with thread creation/scheduling and decreased caching efficiency limit the speed-up on more processors.

3 MOLECULAR DYNAMICS SIMULATION

The Molecular Dynamics method[9] simulates the dynamics of a system of N interacting atoms by temporal integration of the Newton's equations of motion:

$$m_i \frac{d^2 \underline{x}_i}{dt^2} = F_i, \quad i = 1, \ldots, N \tag{1}$$

where $\underline{x}_i$ denote the particle positions and the forces F_i are the negative derivatives of a potential function $U_i(\underline{x}_1, \ldots, \underline{x}_N)$. In the present work, the system of N particles is integrated in time from time level n to $n+1$ using the velocity form of the second-order

accurate, time reversible, Verlet algorithm[10]:

$$\begin{aligned} \underline{x}_i^{(n+1)} &= \underline{x}_i^{(n)} + \delta t \underline{v}_i^{(n)} - \frac{\delta t^2}{2m_i} \nabla U_i^{(n)} \qquad i = 1, \ldots, N \\ \underline{v}_i^{(n+1)} &= \underline{v}_i^{(n)} + \frac{\delta t}{2m_i} \nabla U_i^{(n)} \qquad i = 1, \ldots, N \end{aligned} \tag{2}$$

where $\underline{v}_i$ denotes the velocity of particle i. The Verlet algorithm is a standard temporal integration method in Molecular Dynamics, it has excellent energy conservation properties and it is computationally efficient.

The use of Newton's equation of motion automatically implies the use of classical mechanics to describe the motion of the atoms. At normal temperatures and for most atoms, this use of classical mechanics is justified. However, for intra-molecular atomic motions in poly-atomic molecules, e.g. vibrational motions, quantum mechanical effects need to be included. The vibrational motion can be modelled employing a classical harmonic oscillator, however, with significant restrictions on applicability and validity. The classical oscillator absorbs too much energy in its ground state when compared to the quantum oscillator. The difficulty in modelling the vibrational motion and the fact that the characteristic time-scales of the vibrational motion are much shorter than those for the translational motion, has motivated the Constrained Molecular Dynamics approach for polyatomic molecules, in which the atomic bonds and bond angles are treated as constraints in the equations of motion. A major practical advantage is that in this case a significantly larger time step (typically four times larger) can be used compared to simulations including the oscillatory atomic motion.

The force field in equations (1) and (2) is pair-additive, which implies that all non-bonded forces result from the sum of non-bonded pair interactions. In the present work, the inter-particle force field is modelled by a Lennard-Jones potential:

$$U_{LJ}(r_{ij}) = 4\epsilon_{ij}\Big[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6} \Big] \tag{3}$$

The Lennard-Jones potential combines repulsive and dispersive terms. In the above equation, σ_i as well as ϵ_i are assumed identical for $i = 1, \ldots, N$. Furthermore, a cut-off distance r_c is defined. Therefore, particles within this distance interact through Equation (3), while beyond this range, the contribution to the inter-particle potential is neglected. Here, the cut-off distance is chosen such that $r_c = 2.5\sigma$. Long-range interactions, such as Coulomb forces are not included and the present method only considers short-range particle interactions based on Equation (3). An important characteristic of MD simulations is that the atoms can and will probably undergo large displacements over the duration of the simulation. The important feature from a computational standpoint is that each atom's neighbours within a cut-off distance r_c change as the simulation progresses. Two methods have been widely used to efficiently create a list of each atom's neighbours during the course of a simulation, i.e. the Verlet algorithm and the link-cell algorithm.

The Verlet algorithm introduces a neighbour list which holds all neighbouring particles within an extended cut-off distance $r_c + \delta$, this enables the list to be re-used for a few time steps to calculate all interactions. The choice of δ is problem-dependent.

The link-cell algorithm is used in the present method, with a uniform background mesh defined for the computational domain with cells of side length d, where $d = r_c$ or slightly larger. This reduces the task of finding neighbours of a given particle to checks over 27 cells, i.e. the cell in which the cell is located at its 26 nearest neighbour cells. The link-cell algorithm for short-range particle interactions is sketched in Figure 10.

The link-cell algorithm, as well as the Verlet method, reduce the computational complexity from $O(N^2)$ for a direct computation of all particle interactions to $O(N)$. Highly optimised MD solvers typically employ a combination of the Verlet neighbour list and the link-cell algorithm. In this case, the search space for neighbours within the interaction range can be reduced from $27r_c^3$ to $4/3\pi r_c^3$.

The molecular dynamics method is well-suited for parallelisation since the force calculations and velocity/position can be done simultaneously all particles. Two basic approaches can be discerned: i) particle decomposition, where each processor is assigned a sub-group of the particles, ii) spatial decomposition, where each processor is assigned a portion of the physical simulation domain.

The examples shown in the present work follow the particle decomposition approach, which is well-suited for shared-memory parallelism. For message-passing implementations and for simulations of systems with a large number of particles, the spatial decomposition approach is often preferred[11]. Widely used Molecular Dynamics methods, e.g. GROMACS[12] and NAMD[13], are typically based on the message passing paradigm.

3.1 Implementation

The Molecular Dynamics method discussed here has been recently developed to fit into the multi-disciplinary CFD framework that forms the long-term aim of the current research. An object-oriented design was devised with a high level of modularity and flexibility with the aim to provide the opportunity to prototype different boundary conditions and coupling strategies with different simulation methods, i.e. mesh-based methods. The parallelisation is based on the particle decomposition method, in which the particle potential computation is implemented such that multiple threads cannot simultaneously update the potential of a particle, i.e. synchronisation overhead is avoided. In the object-oriented design, the formation of temporary variables is avoided when operator overloading is employed in parallel regions, since the synchronisations of the memory allocation operations would severely limit the parallel efficiency. An important aspect of the present C++ implementation is the use of classes which create a data structure compatible with C-style for-loops, since OpenMP prevents a direct use of C++ Standard Template Library-style iterators instead of more traditional for/do-loops.

3.2 Parallel performance

The parallel performance was analysed by conducting simulations for a system of initially randomly placed particles interacting through a Lennard-Jones potential. Different problem sizes are considered here, ranging from 625 particles on a $5 \times 5 \times 5$ link-cell mesh to $80,000$ particles on a $20 \times 20 \times 20$ mesh. For all test cases, the average number of particles per cell is either 5 or 10. For problems with on average 5 particles per cell, on average 135 particles will be tested for being within the interaction range, with on average 21 particles within this range. Periodic boundary conditions were imposed on the domain. The simulations were run sufficiently long for particles to move from one cell to the next, i.e. significant changes in the neighbour lists occur during the simulation. The parallel performance was again tested on the 4-way Opteron SMP and the NEC TX-7 cc-NUMA machines at HLRS. The obtained results are shown in Table 3, showing excellent speed-up for the more realistically sized larger problems.

Table 3: Parallel 3D MD simulations. Scaling for different problems sizes.

	4-way Opteron		NEC TX-7				
nproc.	625 (5^3)	5000 (10^3)	625 (5^3)	5000 (10^3)	10000 (10^3)	40000 (20^3)	80000 (20^3)
2	1.94	1.98	1.80	1.86	1.93	1.98	1.97
4	3.63	3.81	3.16	3.57	3.78	3.91	3.91
6	-	-	4.43	5.20	5.58	5.81	5.83
8	-	-	5.65	6.79	7.41	7.70	7.76

4 SUMMARY OF CONCLUSIONS AND FUTURE WORK

The shared-memory parallelisation of the multi-block CFD solver is based on a loop-parallelisation of loops over the grid blocks. The parallel performance of this implementation was found to be highly dependent on the load (im)balance that results from this decomposition. For realistic aerospace configurations, i.e. with a significant variation in block dimensions, this coarse-grained OpenMP implementation does result in a significantly higher load imbalance as compared to the MPI implementation. Furthermore, it was found that the parallel efficiency does significantly depend on the loop-scheduling options chosen and that the most appropriate option depends on the characteristics of the used multi-block mesh. The parallel performance assessment showed that a further, finer-grain optimisation is required to improve the parallel speed-up of the solver for 4 or more processors. It is expected that as the problem size increases the overhead due to the thread creation will decrease leading to better efficiencies for the shared-memory version of the CFD solver. Further optimisations, including improvements to the data cache util-

isation, are also underway. A further improvement can be obtained by a re-numbering of the grid blocks based on the block dimensions. In this case, the OpenMP implementation could reach similar levels of lad balance as the MPI implementation.

The development of the parallel object-oriented molecular dynamics method shows that the method is well-suited for shared-memory parallelisation with OpenMP. This parallel efficiency is achieved using an implementation based on the particle decomposition method. The object-oriented design accounts for shared-memory parallelisation requirements and limitations in a number of ways. First, the particle potential computation was implemented such that multiple threads cannot simultaneously update the potential of a particle, i.e. synchronisation overhead is avoided. Furthermore, classes were used which create a data structure amenable for loop parallelisation with OpenMP. Finally, the synchronisation overhead associated with dynamic memory allocation within parallel regions was avoided by suppressing the formation of temporary variables whenever operator overloading was used.

At present, various parallelisation strategies (message-passing and shared-memory) are being investigated for multi-disciplinary CFD methods. The present work showed that the molecular dynamics method, illustrative of particle-based solution methods in a multi-disciplinary CFD method, has an excellent parallel performance for the considered shared-memory implementation. The block-structured finite-volume CFD method considered here was originally designed to employ the message passing paradigm for parallelisation. Based on this data structure, the effectiveness of the OpenMP-based implementation was shown to be limited to around 4 CPUs. For the future development of the mesh-based solution method in the multi-disciplinary CFD framework, the aim is to create a parallel message-passing implementation (based on the MPI standard) which allows a fine-grained shared-memory OpenMP formulation to be used when hybrid shared/distributed memory computers are used.

Acknowledgements

The work on the shared-memory parallelisation of the CFD solver was carried out during a one-month stay at the Institut für Gasdynamik und Aerodynamik (IAG), Universität Stuttgart and the Bundeshöchstleistungrechenzentrum Stuttgart (HLRS), sponsored through the EC-funded project HPC-Europa, contract number 506079. The authors would like to thank Rolf Rabenseifner, Rainer Keller (HLRS), Manuell Kessler and Marcus Dietz (IAG) for the support during this project.

REFERENCES

[1] J. Hoeflinger, P. Alavilli, T. Jackson, and B. Kuhn. Producing Scalable Performance with OpenMP: Experiments with Two CFD Applications. *Parallel Computing*, 27:391–413, 2001.

[2] D.S. Balsara and C.D. Norton. Highly Parallel Structured Adaptive Mesh Refinement

using Parallel Language-Based Approaches. *Parallel Computing*, 27:37–70, 2001.

[3] K.J. Badcock, B.E. Richards, and M.A. Woodgate. Elements of computational fluid dynamics on block structured grids using implicit solvers. *Progress in Aerospace Sciences*, 36:351–392, 2000.

[4] R. Steijl, G.N. Barakos, and K.J. Badcock. A Framework for CFD Analysis of Helicopter Rotors in Hover and Forward Flight. Accepted for publication in Int. J. Numer. Meth. Fluids, August, 2005.

[5] S. Osher and S. Chakravarthy. Upwind schemes and boundary conditions with applications to euler equations in general geometries. *J. Computational Physics*, 50:447–481, 1983.

[6] A. Jameson. Time Dependent Calculations Using Multigrid, with Applications to Unsteady Flows past Airfoils and Wings. AIAA Paper 1991-1596, 10th Computational Fluid Dynamics Conference , Honolulu, Hawai, June 24-26, 1991.

[7] F.X. Caradonna and C. Tung. Experimental and analytical studies of a model helicopter rotor in hover. Technical Report TM-81232, NASA, 1981.

[8] K.J. Schultz, W. Splettstösser, B. Junker, W. Wagner, and G. et al. Arnaud. A parametric windtunnel test on rotorcraft aerodynamics and aeroacoustics (helishape) - test procedures and representative results. *Aeronautical Journal*, 101:143–154, 1997.

[9] M.P. Allen and D.J. Tildesly. *Computer Simulation of Liquids*. Clarendon Press, Oxford, 1987.

[10] L. Verlet. Computer Experiments of Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. *Physical Review*, 159:98, 1967.

[11] S. Plimpton. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *J. Comput. Phys.*, 117:1–19, 1995.

[12] D. van der Spoel, E. Lindahl, B. Hess, A.R. van Buuren, E. Apol, P.J. Meulenhoff, P.J. Tieleman, A.L.T.M. Sijbers, K.A. Feenstra, R. van Drunen, and H.J.C. Berendsen. GROMACS User Manual, Version 3.2. Technical report, University of Groningen, GROMACS Development Team, www.gromacs.org, 2004.

[13] L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadrajan, and K. Schulten. NAMD2: Greater Scalability for Parallel Molecular Dynamics. *J. Comput. Phys.*, 151:283–312, 1999.

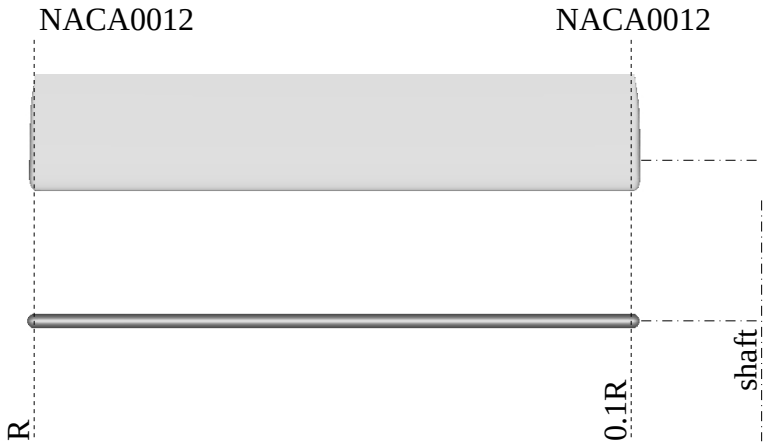

Figure 1: Planform geometry of Caradonna-Tung[7] model rotor.

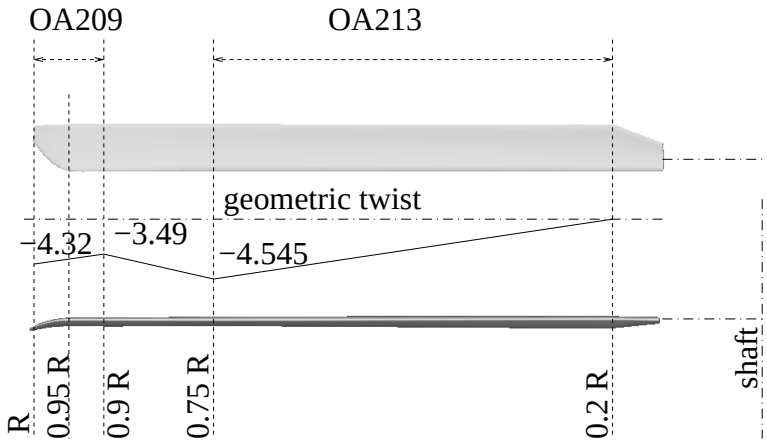

Figure 2: Plan geometry and blade-twist distribution of ONERA 7AD1[8] model rotor.

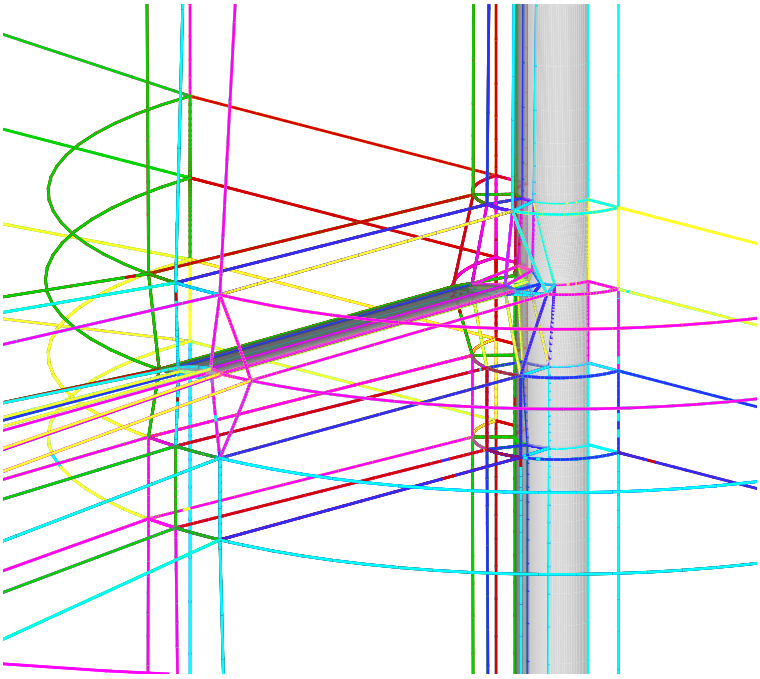

Figure 3: Multi-block topology for Caradonna-Tung rotor, 74 blocks.

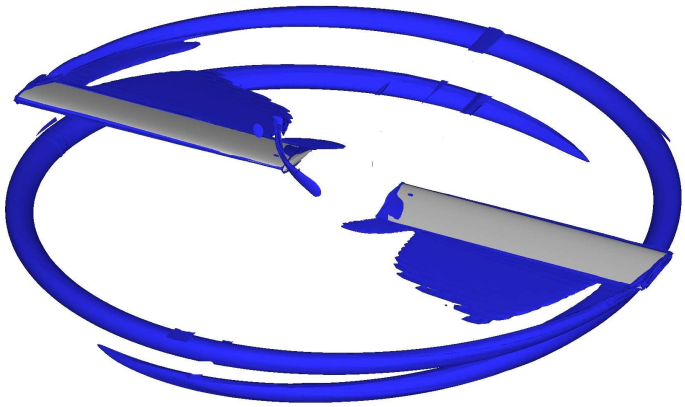

Figure 4: Wake of hovering Caradonna-Tung rotor

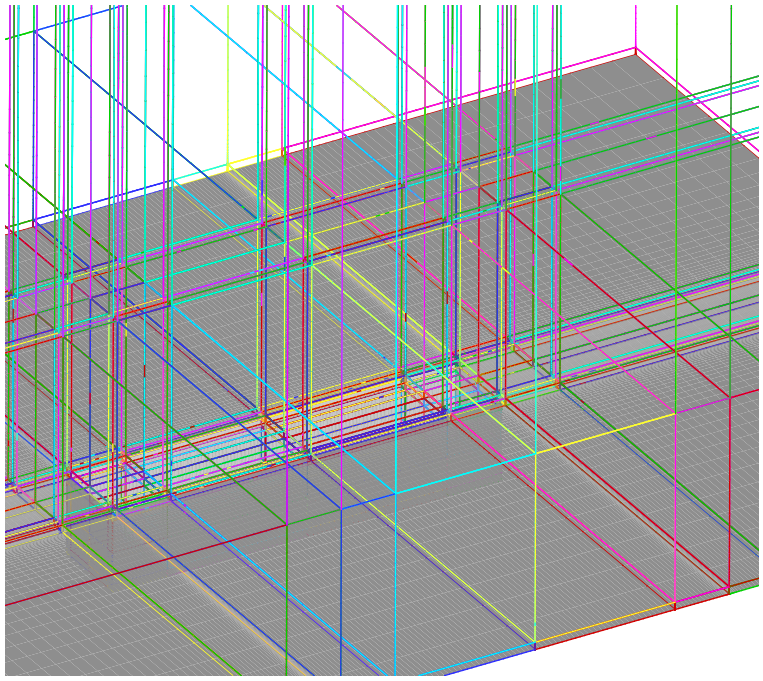

Figure 5: Multi-block topology for LES simulation of transonic cavity flow. Length/depth ratio L/D=5, width/depth ratio W/D=1. 256 blocks, $8,000,000$ grid points.

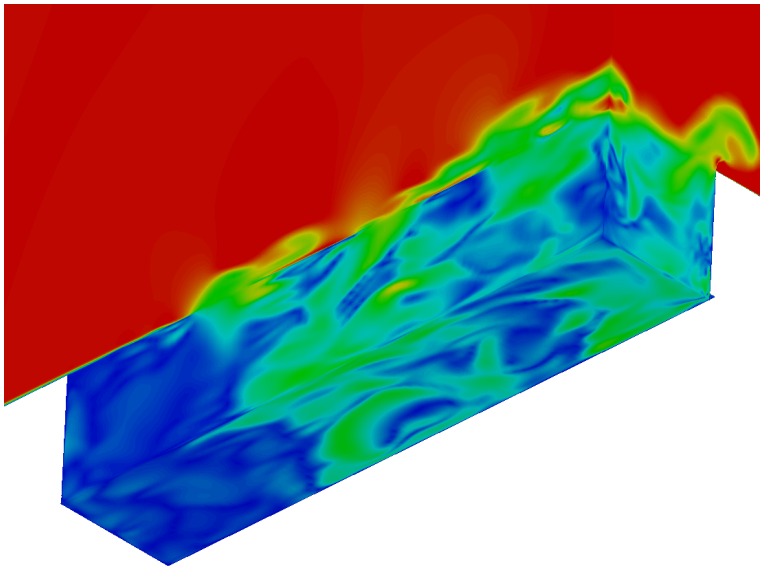

Figure 6: Flow visualisation of the flow features inside the 3D, L/D=5, W/D=1 cavity modelled using LES. Free-stream Mach number 0.85, Reynolds number is one million based on the cavity length. Shown are instantaneous Mach number contours.

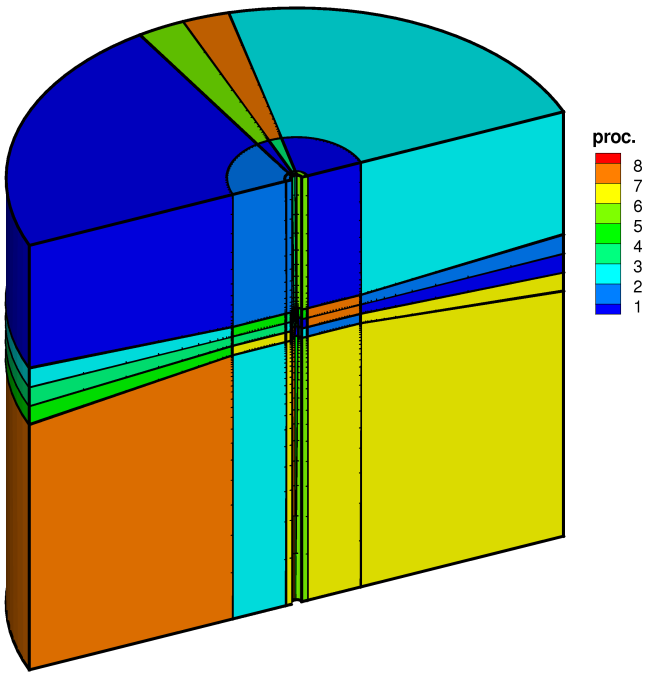

Figure 7: Block assignment to processors. Caradonna-Tung rotor, 74-block grid. Simulation on 8 CPUs

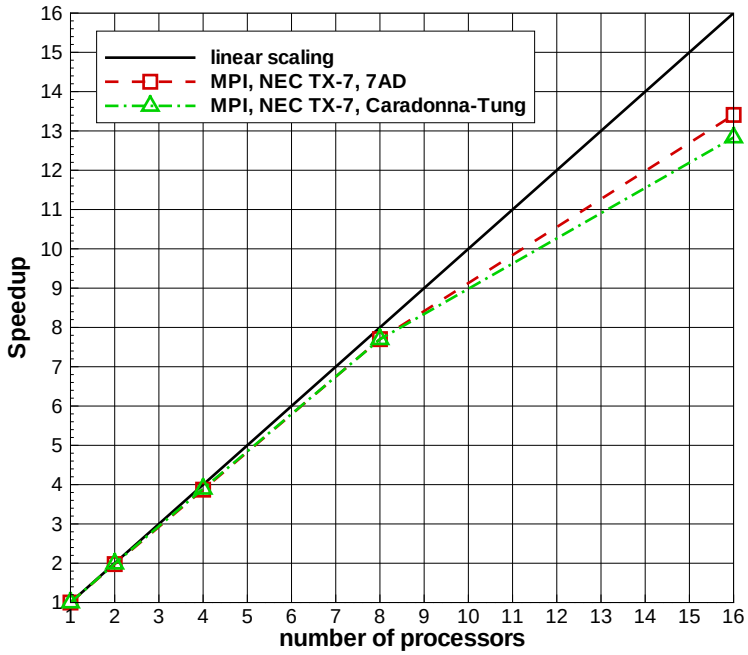

Figure 8: Scaling for MPI implementation. NEC TX-7 cc-NUMA shared-memory multiprocessor. Considered are two hovering rotors, the 7AD rotor with 600, 000 grid points in 98 grid-blocks and the Caradonna-Tung rotor with 1, 200, 000 grid points in 74 grid-blocks.

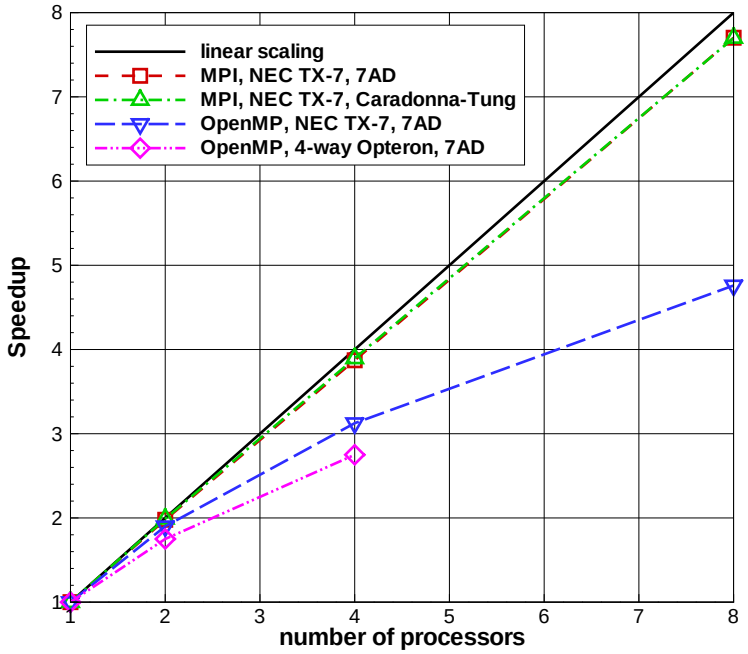

Figure 9: Scaling for OpenMP implementation. NEC TX-7 cc-NUMA and 4-way Opteron. Considered are two hovering rotors, the 7AD rotor with 600, 000 grid points in 98 grid-blocks and the Caradonna-Tung rotor with 1, 200, 000 grid points in 74 grid-blocks.

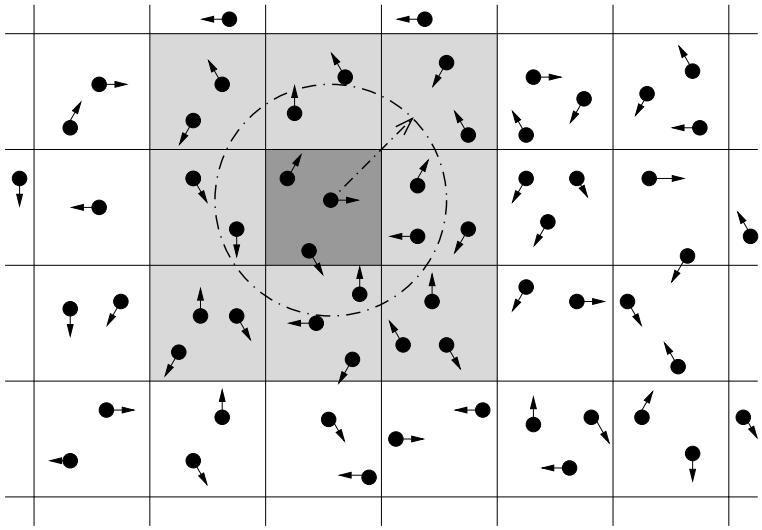

Figure 10: Link-cell algorithm for computation of short-range particle interactions. During a simulation, the particles move through equally-spaced Cartesian cells. For the considered particle (located in the dark shaded cell), particles within the interaction rage (indicated with circle) are located within this cell and its 26 nearest neighbour cells (3D, 8 nearest neighbours in 2D).